KONSTRUKČNÍ ELEKTRONIKA A Radio

ROČNÍK XII/2007. ČÍSLO 1

Amatérské RADIO

ŘADA B - PRO KONSTRUKTÉRY

ROČNÍK LVI/2007. ČÍSLO 1

V TOMTO SEŠITĚ

KONSTRUKČNÍ ELEKTRONIKA A RADIO

Vydavatel: AMARO spol. s r. o.
Redakce: Zborovská 27, 150 00 Praha 5, tel.: 2 57 31 73 11, tel./fax: 2 57 31 73 10.
Šéfredaktor ing. Josef Kellner, sekretářka redakce Eva Marková, tel. 2 57 31 73 14.
Ročně vychází 6 čísel. Cena výtisku 36 Kč.
Rozšiřuje PNS a. s., Transpress spol. s r. o., Mediaprint & Kapa a soukromí distributoři.
Předplatné v ČR zajišťuje **Amaro** spol. s r. o. - Michaela Hrdličková, Hana Merglová (Zborovská 27, 150 00 Praha 5, tel./fax: 2 57 31 73 13, 2 57 31 73 12. Distribuci pro předplatitele také provádí v zastoupení vydavatele společnost Mediaservis s. r. o., Abocentrum, Moravské náměstí 12D, P. O. BOX 351, 659 51 Brno; tel: 5 4123 3232; fax: 5 4161 6160; abocentrum@mediaservis.cz; reklamace - tel.: 800 800 890.
Objednávky a predplatné v Slovenskej republike vybavuje MAGNET-PRESS Slovakia s. r. o., Šustekova 8, 851 04 Bratislava,
tel.: 00421 2 / 6720 1931 - 33
email: predplatne@press.sk ; www.press.sk
Podávání novinových zásilek povoleno Českou poštou - ředitelstvím OZ Praha (č.j. nov 6005/96 ze dne 9. 1. 1996).
Inzerci v ČR přijímá redakce, Zborovská 27, 150 00 Praha 5, tel.: 2 57 31 73 11, tel./fax: 2 57 31 73 10.
Inzerci v SR vyřizuje MAGNET-PRESS Slovakia s. r. o., Šustekova 8, 851 04 Bratislava,
tel.: 00421 2 / 6720 1931 - 33 ; www.press.sk
Za původnost a správnost příspěvků odpovídá autor (platí i pro inzerci). Nevyžádané rukopisy nevracíme.
http://www.aradio.cz; *E-mail*: pe@aradio.cz
ISSN 1211-3557, MK ČR E 7443

Z dějin vědy a techniky

Historie elektřiny a magnetizmu

Pozvánka do muzea - expozice elektrické měřicí techniky

Když jsme v letošním roce sbírali materiál pro reportáž z veletrhu AMPÉR, nemohli jsme opominout expozici firmy Metra Blansko a. s., kde jsme mimo obvyklých informací o nejnovějších výrobcích (viz AR 5/06 str. 39) tentokrát získali také prospekt a pozvání k návštěvě expozice měřicí techniky, kde jsou vystaveny měřicí přístroje z produkce Metry Blansko. Je umístěna v Muzeu Blansko.

Když „dozrál čas“, několik telefonátů mi potvrdilo, že taková expozice skutečně existuje, a umožnilo mi na dohodnutý den a čas získat i zasvěceného průvodce, který se na jejím vzniku podílel.

Několik seniorů - bývalých zaměstnanců firmy, pro které byla práce na speciální měřicí technice nejen zaměstnáním, ale i koníčkem, založilo v roce 1996 „Společnost pro identitu měřicí techniky - SPRIMT“. Ti tuto expozici, v naší republice ojedinělou, neúnavně a nezištně sestavili, a nyní ji zdokonalují. Jsou mezi nimi i zahraniční členové, jako jedenadevadesátiletý Čechošvýcar. Expozice je postupně bezplatně převáděna do státních muzejních fondů. Mým průvodcem byl jeden ze členů SPRIMTu, p. Miroslav Starycha. Vše vyšlo podle dohodnutého plánu, takže zakrátko po vzájemném představení jsem již vstřebával hutné informace o vzniku továrny, o jejím zakladateli, o činnosti SPRIMTu, a nakonec jsem si také prohlédl vystavené přístroje. Řada z nich mi byla důvěrně známá - z fyzikálního kabinetu na gymnáziu, ty novější z laboratoří na VŠ a v zaměstnání, některé dodnes sám doma používám. Střední a starší generace jistě opatruje některý z AVOMETů, mladí některý přístroj z řady PU.

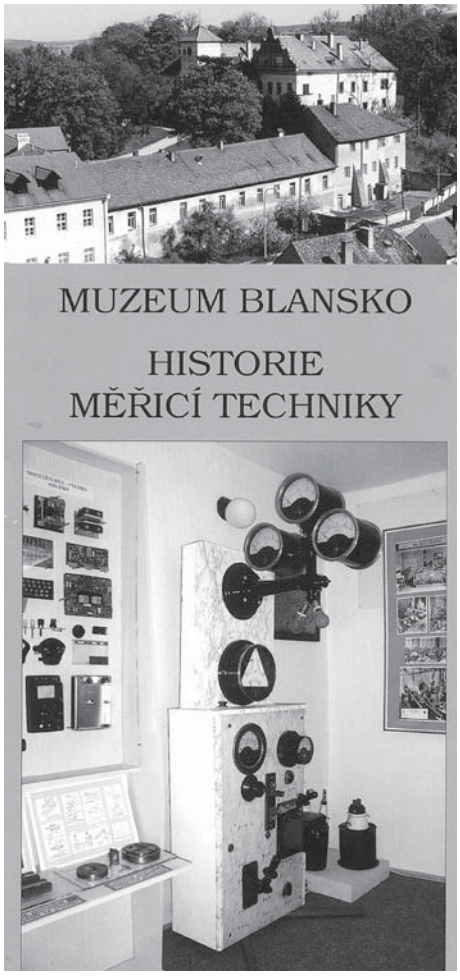
Název METRA Blansko dostal podnik, který založil již v roce 1911 Ing. Erich Roučka, v roce 1948. Původně to byla jediná továrna na výrobu elektrotechnických měřicích přístrojů ve střední a východní Evropě a nesla v názvu jméno zakladatele - Roučka. V roce 1934 přešla do vlastnictví Roberta Sochora, poněvadž Roučka si mezitím vybudoval další továrnu, tentokráte na regulační techniku pro parní stroje, v Brně - Slatině. Jak předválečné výrobky, tak i ty, které nesly již typické

Znak Metry Blansko

označení - stylizované písmeno omega se šipkou, které bylo možno číst i jako M (Metra), vynikaly vždy precizností a spolehlivostí. Řada z nich, přestože byly vyrobeny ve 30. letech minulého století, je dodnes funkční.

Metra Blansko a. s. dnes patří do konsorcia více firem s názvem APOS s. r. o., ale jednotlivé firmy vystupují samostatně. Některé historické exponáty jsou vystaveny v její vstupní hale. V minulých letech byl vytvořen také velký podnik na výrobu elektrické měřicí techniky mimo území ČR - Metra Ukrajina.

Samotná expozice je rozdělena na několik částí a bohužel trpí jedním zá-



kladním neduhem - tím je nedostatek větších prostor. Proto některé větší exponáty (celkem je jich asi 1000 kusů - např. měřicí stoly) jsou stále umístěny v depozitáři.

První část expozice je věnována zakladateli továrny ing E. Roučkovi a několika historickým exponátům. V dalších částech najdeme laboratorní přístroje a odporové a napěťové normály. Pro každého, kdo si nedovede představit, jak „střeva“ každého měřicího přístroje vypadají, jsou na kruhové podložce umístěny jednotlivé díly systému a je možné si je prohlédnout pod mikroskopem. Starší generace fotografů-amatérů tam najde populární „Expozimet“ z 50. let, stále se vyrábí lokomotivní tachografy, modernizované registrační přístroje apod. Ale najdete tam řadu přístrojů, o kterých jste určitě ani neměli potuchu, že je Metra vyráběla či vyrábí - např. laserový interferometr k měření délek s přesností desetiny mikronu, pravděpodobně první naše digitální měřidla a jiné speciální přístroje. Dnešní revizní technici se téměř neobejdou bez moderních sdružených přístrojů řady PU, i údržba telefonních linek používá speciální přístroje, které pro ně Metra Blansko vyráběla.

Nakonec - několik doprovodných fotografií (viz druhá strana obálky) napoví asi více, než by to dokázal další text.

Práci skupiny fandů, která se o expozici stará, je nutné vyzvednout. Je jen škoda, že jiné továrny nepečují podobným způsobem o výrobky, které je kdysi proslavily - vzpomeňte jen na jednotlivé závody TESLA, z nichž mnohé dnes ani neexistují. Čtenářům, pokud pojedou přes Blansko, doporučuji expozici měřicí techniky navštívit - vedle návštěvy Moravského krasu je to další možný impuls k návštěvě Blanska, zejména pro techniky. Použijete-li k návštěvě vlakové spojení, vystupte na zastávce Blansko-město, odtamtud je k muzeu jen asi 3 minuty chůze. Snad se v budoucnu podaří pro tuto expozici získat i důstojnější prostory, aby mimo vlastních přístrojů byla k dispozici pro zájemce případně i jejich technická dokumentace.

QX

Erich Roučka

V plejádě významných zahraničních osobností, jejichž význam pro technický pokrok je nesporný, a jejichž životopisy se již dlouho objevují na těchto stránkách, se občas objeví i jméno české (Diviš, Kaplan, Křižík, Svoboda) či slovenské (Murgaš).

Nejinak je tomu i dnes. Představíme vám vysoce technicky erudovaného a podnikatelsky zdatného inženýra Ericha Roučku, který již ve svých 22 letech uvedl v Blansku do provozu svou první továrnu. Narodil se v roce 1888 ve Velkém Meziříčí a s podnikáním začal ihned po studiích. Jeho továrna byla ve střední Evropě první, která začala s výrobou elektrických měřicích přístrojů. Továrna přes pozdější změny majitelů i změny názvů až po dnešní akciovou společnost Metra Blansko si dodnes uchovala punc vysoké kvality a spolehlivosti výrobků a přes ostrý konkurenční boj si také dodnes zachovala schopnost konkurovat i na cizích trzích a prosperovat.

Náš průmysl byl do té doby zaměřen spíše na klasické „těžké“ strojírenství a tento obor byl něčím zcela novým. Roučka pilně studoval vše, co se mu zdálo být perspektivní a tak také pronikl do problematiky řídicí a regulační techniky pro vysokotlaké parní kotle. Aby se seznámil se zkušenostmi, které měli v daném oboru v zahraničí, neváhal cestovat. Tak se také dostal do Ameriky, kde se setkal dokonce i s Edisonem, se kterým vedl diskuse o budoucnosti odvětví, ve kterých se chtěl angažovat a obeznámil se s patentovou problematikou.

V roce 1929 začal budovat v Brně - Slatině další továrnu, jejíž výrobním programem byly hlavně parní kotle, a její součástí byla i slévárna (dnes a. s. Roučka - Slatina). Hodně četl, a když se seznámil s čapkovým literárním dílem, vyžádal si od Karla Čapka souhlas, aby mohl některé automatické řídicí prvky které vyráběl, nazývat E. R. Robot (Erich Roučka Robot). Tyto mechanizmy byly u nás něčím zcela novým a na jejich konstrukci získal řadu patentů. Získaly i mnoho zahraničních ocenění, neboť jejich uplatnění znamenalo velké úspory paliv a měly také ekologický přínos. Požadavky odběratelů rostly a tak se snažil svou továrnu v Blansku rozšiřovat, ale setkal se s nepochopením radních, kteří mu další rozšíření továrny nepovolili. Proto v roce 1934 prodal továrnu podnikateli Robertu Sochorovi a soustředil se plně na výrobu v Brně. Jeho další patenty se týkaly využití levného hnědého uhlí pro vytápění větších objektů a jeho kotelny se postupně zaváděly v celé ČSR. Ve slévárně brněnské továrny zavedl např. také výrobu vysokojakostních ocelolitin z odpadů železného šrotu.

Vysoké pracovní nasazení a časté zahraniční cesty se však podepsaly jak na jeho rodinném životě, tak na zdraví. Manželka, se kterou měl tři děti (jeho dcera dodnes žije v Brně), se s ním nakonec rozvedla, a on sám vážně onemocněl totálním vyčerpáním tak, že mu lékaři prakticky nedokázali pomoci. Tehdy začal studovat životosprávu etnických skupin, které se vyznačovaly dlouhověkostí, a sám na sobě začal experimentovat. Tím se dostal z krize z vyčerpání, a aby zkušenosti, které tak získal, využil, psal různá pojednání o správné životosprávě. V Brně založil dokonce laboratoř, která se zabývala tím, co dnes nazýváme gerontologií. Ve svých pracích upozorňoval na rizika spojená s nevhodným stravováním (včetně vlivu na vznik rakoviny) a sám se poznaných zásad držel, což mu přineslo další životní úspěch - dožil se 98 let a byl až do konce života aktivní.

První stánka Roučkova katalogu z 20. let 20. století

O jeho všestranných schopnostech svědčí i to, že založil sanatorium, kde se uplatňovaly při léčení jeho poznatky a zdravotní zásady.

Po válce byl jeho podnik znárodněn a on sám byl penzionován. Nesměl se dokonce podílet ani na výzkumu, přestože jeho výsledky chtěl předávat státu. Vyměřená penze byla tak malá, že z ní nemohl uživit ani svou druhou ženu. Nakonec našel částečné uplatnění ve Výzkumném ústavu energetickém, kde vyvinul automatický regulátor pro vysokotlaké kotle, u kterých se jako palivo používalo práškové uhlí. Jenže měl oponenty, kteří prosadili dovoz regulátorů ze zahraničí, a ty, jak se nakonec ukázalo, byly velmi nespolehlivé. To jej natolik zklamalo, že ve svých 70 letech emigroval do Německa, kde ještě mnoho let pracoval jako technik u firmy AEG. Ani v posledních více než deseti letech života, kdy žil v ústraní u Norimberku, nezahálel - studoval nové medicínské poznatky a v malé laboratoři experimentoval. Němečtí lékaři, kteří se zabývají alternativní medicínou, považují jeho poznatky z té doby za velmi významné a jeho samotného považují za biochemika. Roučka zemřel v roce 1986. In memoriam byl jmenován čestným občanem města Blanska a Metra Blansko nyní uděluje svým zasloužilým pracovníkům Cenu Ericha Roučky.

QX

Podklady ke zpracování tohoto článku poskytl laskavě pan M. Starycha ze sdružení SPRIMT.

Literatura

[1] Elektrické měřící přístroje - katalog Roučkovy továrny (20. léta).

[2] Aperiodické precisní měřící přístroje - katalog Roučkovy továrny z r. 1917.

[3] Přenosné univerzální měřicí přístroje řady PU. Technomat, České Budějovice 1971.

[4] Informační prospekty Muzea Blansko.

PROGRAMOVÁNÍ A APLIKACE MIKROŘADIČE ATtiny2313 (2. díl)

Ing. David Matoušek

Tento článek navazuje na předchozí článek o mikrořadiči Atmel ATtiny 2313, který byl uveřejněn v KE5/2006. Proto je číslování kapitol vedeno dále - první kapitola má tedy číslo 9.

Text je, podobně jako v předchozím dílu, sestaven bez složitých teoretických statí. Proto v něm najdete především podrobně komentované příklady. Postup jejich realizace je vyložen do co nejmenších detailů tak, aby je mohl dobře pochopit i začátečník a nemusel přitom hledat vysvětlení v další literatuře.

Úvod

Devátá kapitola popisuje analogový komparátor vestavěný do mikrořadiče, v příkladu je použit pro měření odporu. Naměřený odpor se zobrazuje na displeji LCD.

Desátá kapitola popisuje jednotku USART, kterou lze využít např. pro komunikaci mikrořadiče s osobním počítačem. Příklady představují základy komunikace s PC a dále vytvoření generátoru impulsů, který je řízen počítačem.

V jedenácté kapitole se seznámíme s jednotkou USI, což je dvou/třídrátová sběrnice. Praktické použití bude předvedeno na konstrukci sériově řízeného třímístného displeje.

Dvanáctá kapitola shrnuje paměťové prostory a registry dostupné v mikrořadiči ATtiny2313.

Ve třináctě kapitole najdete podrobně komentovaný výklad instrukčního souboru mikrořadiče ATtiny2313 včetně mnoha příkladů.

Čtrnáctá kapitola popisuje zbývající rysy mikrořadiče, jako jsou: zdroje synchronizace, zdroje resetu a propojky (fuses).

Poslední (patnáctá) kapitola popisuje konstrukci dvou užitečných přípravků určených pro USB.

Přípravek FT232TST představuje emulátor sériového portu, který nahradí sériový port počítače a umožní tak připojování jednotky USART přímo na USB.

Přípravek USB2313KIT představuje vývojový kit mikrořadiče ATtiny2313, který se řídí a napájí přímo z počítače přes sběrnici USB. Kit má malé rozměry a oceníme ho hlavně pro jeho schopnost získat napájení přímo ze sběrnice USB.

9. Vestavěný analogový komparátor

Analogový komparátor umožňuje porovnávat velikosti dvou vstupních napětí. Lze jej použít např. pro realizaci regulačních úloh nebo pro měření fyzikálních veličin.

Schéma analogového komparátoru je na obr. 9.1. Komparátor porovnává napětí na vývodech svého neinvertujícího vstupu (*U*+) a invertujícího vstupu (*U*-).

Je-li $U+ \leq U-$, je výstup komparátoru v úrovni „log. 0“. Je-li $U+ > U-$, je výstup komparátoru v úrovni „log. 1“. Výstup komparátoru je představován bitem **ACO** z registru **ACSR**.

Výstup komparátoru lze, kromě čtení jeho logické hodnoty pomocí bitu **ACO**, použít pro generování zvláštního přerušení. Tento výstup lze také použít k záchytu stavu čítače/časovače 1 v jeho záchytném registru. Spouštěcí událost odpovídá náběžné nebo sestupné hraně výstupu komparátoru, spouštění je možné i při změně logické hodnoty výstupu komparátoru.

9.1. Registr DIDR

Funkci komparátoru ovlivňují bity **AIN1D** a **AIN0D** z registru **DIDR** (viz obr. 9.2).

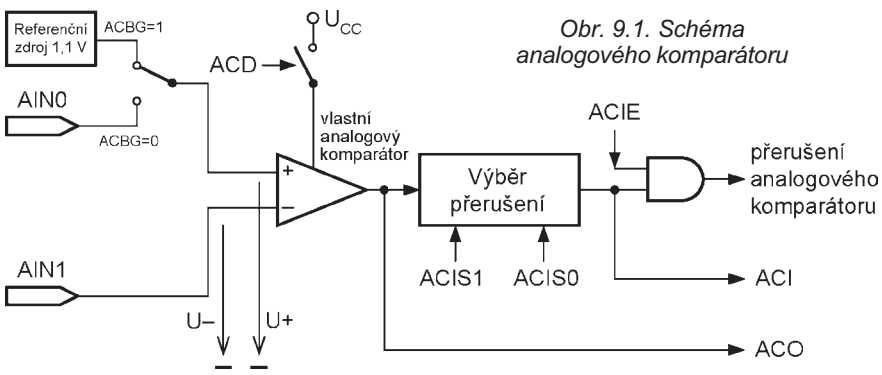
• **AIN1D, AIN0D - odstavení digitálního vstupního bufferu.** Nastavením těchto bitů odpojíme digitální vstupní buffer příslušného vývodu. Při použití těchto vývodů pro analogový komparátor totiž není digitální buffer nutný a zároveň snížíme odběr.

9.2. Registr ACSR

Analogový komparátor je řízen registrem **ACSR**. Tento registr obsahuje jak řídicí bity, tak i stavové příznaky analogového komparátoru (obr. 9.3).

Význam jednotlivých bitů:

• **ACD - odstavení analogového komparátoru.** Je-li **ACD = 1**, je odpojeno napájení pro analogový komparátor (viz obr. 9.1). To výrazně sníží spotře-



Obr. 9.1. Schéma analogového komparátoru

*Obr. 9.2. Registr **DIDR***

Bit	7	6	5	4	3	2	1	0
DIDR	–	–	–	–	–	–	AIN1D	AIN0D
Čtení/zápis	R	R	R	R	R	R	R/W	R/W
Výchozí hodnota	0	0	0	0	0	0	0	0

*Obr. 9.3. Registr **ACSR***

Bit	7	6	5	4	3	2	1	0
ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
Čtení/zápis	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	?	0	0	0	0	0

bu mikrokontroléru především v režimech spánku. Je-li **ACD = 0**, je napájení analogového komparátoru připojeno. Pokud se má změnit stav bitu **ACD**, je vhodné vynulovat bit **ACIE**. Jinak může být generováno nežádoucí (falešné) přerušení.

• **ACBG - připojení referenčního zdroje na neinvertující vstup**. Je-li **ACBG = 1**, je na neinvertující vstup analogového komparátoru připojen vestavěný zdroj referenčního napětí (se jmenovitou velikostí 1,1 V). Je-li **ACBG = 0**, je neinvertující vstup analogového komparátoru napojen na vývod **AIN0**.

• **ACO - výstup analogového komparátoru.** Tento bit je synchronizován a přímo napojen na výstup analogového komparátoru (viz obr. 9.1). Synchronizace způsobuje zpoždění 1 až 2 hodinové cykly.

• **ACI - příznak přerušení analogového komparátoru. ACI** je nastaven, když výstup analogového komparátoru způsobí spouštěcí událost definovanou bity **ACIS1, ACIS0** (viz tab. 9.1). Je-li **ACIE = 1** a současně **I = 1**, je aktivováno přerušení od analogového komparátoru. Příznak **ACI** se hardwarově vynuluje vykonáním příslušné rutiny obsluhy přerušení. Příznak **ACI** lze nulovat i programově zápisem **1** do **ACI**.

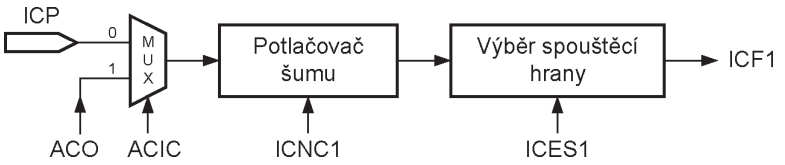
• **ACIE - povolení přerušení analogového komparátoru.** Tento bit povoluje (**ACIE = 1**) nebo zakazuje (**ACIE = 0**) přerušení analogového komparátoru.

• **ACIC - povolení záchytu analogovým komparátorem.** Je-li **ACIC = 1**, je spouštěcí vstup záchytného registru čítače/časovače 1 připojen na výstup analogového komparátoru (viz obr. 9.4). Je-li **ACIC = 0**, je spouštěcí vstup záchytného registru čítače/časovače 1 napojen na vnější vývod **ICP**. Z obr. 9.4 je zřejmé, že signál z výstupu komparátoru (**ACO**) prochází přes potlačovač šumu podobně, jako signál z vnějšího vývodu **ICP**. Pro generování přerušení, které se má aktivovat záchytem pomocí analogového komparátoru, je nutno nastavit bit **TICIE1** v registru **TIMSK**.

• **ACIS1, ACIS0 - výběr přerušení analogového komparátoru.** Tyto bity určují, která událost analogového komparátoru spouští jeho přerušení (viz tab. 9.1). Pokud se mají změnit bity **ACIS1, ACIS0**, je vhodné vynulovat bit **ACIE**. Jinak může vzniknout nežádoucí (falešné) přerušení.

*Tab 9.1. Význam bitů **ACIS1** a **ACIS0***

ACIS1	ACIS0	Režim přerušení
0	0	přerušení při změně stavu výstupu **ACO** (toggle)
0	1	*vyhrazeno*
1	0	přerušení při sestupné hraně výstupu **ACO**
1	1	přerušení při náběžné hraně výstupu **ACO**



Obr. 9.4. Zapojení spouštěcího vstupu záchytného registru čítače/časovače 1

9.3. Přípravek ATRX

Jako praktický doklad toho, že analogový komparátor lze používat pro měření různých fyzikálních veličin, si předvedeme přípravek **ATRX**. Přípravek umožňuje měřit elektrický odpor a případně i kapacitu. Schéma přípravku je na obr. 9.5.

Na invertující vstup komparátoru je pomocí odporového děliče tvořeného rezistory RN1 a RN2 přivedeno napětí odvozené z napájecího napětí. Napětí na invertujícím vstupu je filtrováno kondenzátorem CF.

Na neinvertující vstup komparátoru je připojen obvod *RC*. Kapacitu kondenzátoru CN chápeme jako známou veličinu a měříme odpor rezistoru připojeného na svorky RX. Rezistor RO je ochranný (má minimální odpor).

Bitem D2 je ovládán tranzistor T1, kterým je vybíjen kondenzátor CN.

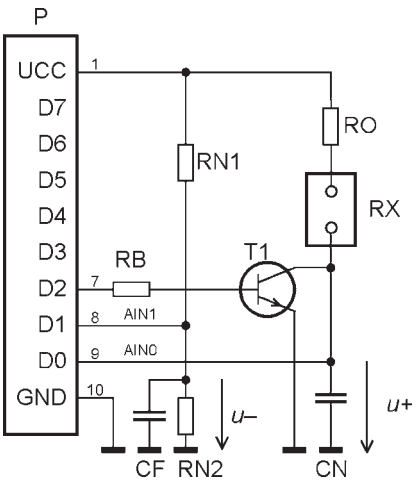
Měření odporu je realizováno programem **PROG_18.ASM**.

Program bude pracovat tak, že nejdříve se sepnutím tranzistoru T1 vybije kondenzátor CN. Poté se nechá kondenzátor CN nabíjet a čítač/časovač 1 současně čítá impulsy. Přitom napětí na kondenzátoru CN je vlastně napětím *U+* na neinvertujícím vstupu komparátoru. Plynule narůstající napětím *U+* na neinvertujícím vstupu je porovnáváno s napětím *U-* na invertujícím vstupu komparátoru. Při dosažení shody *U+* = *U-* je čítání ukončeno a zobrazí se výsledek.

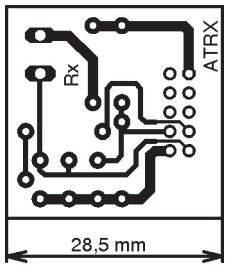
Uvedený algoritmus měření se trvale opakuje.

Doba nabíjení (tn) a tím i počet načítaných impulsů odpovídá přímo úměrně součinu RX×CN (RO zanedbáme). Výsledek není závislý na velikosti napájecího napětí.

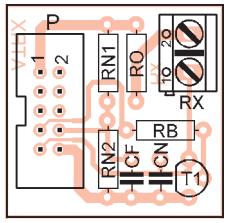
Výkresy potřebné pro výrobu desky s plošnými spoji najdete na obr. 9.6 a obr. 9.7.



*Obr. 9.5. Schéma přípravku **ATRX** (nahoře) a časový průběh nabíjení kondenzátoru CN (dole)*



*Obr. 9.6. Obrazec plošných spojů přípravku **ATRX** (měř.: 1 : 1)*



*Obr. 9.7. Rozmístění součástek na desce přípravku **ATRX***

Seznam součástek pro ATRX

(cena asi 20 Kč)

RB	1 kΩ/1 %/0,6 W
RN1	27 kΩ/1 %/0,6 W
RN2	200 kΩ/1 %/0,6 W
RO	100 Ω/1 %/0,6 W
CF	100 nF/63 V, keramický
CN	100 nF/63 V, keramický
T1	BC547B
P	vidlice MLW10G
RX	svorkovnice ARK500/2

deska s plošnými spoji ATRX

9.4. Příklad 18 - měření odporu

V tomto příkladu bude analogový komparátor použit pro měření odporu.

Zadání: Připojte k vývojovému kitu **SDK2313** (viz KE 5/2006) na port B přípravek **ATRX** a na port D přípravek **ATLCDTX2** (viz KE 5/2006).

Pomocí analogového komparátoru měřte odpor RX.

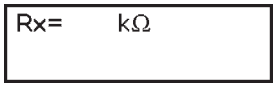
Změřený údaj zobrazte desítkově na displeji přípravku **ATLCDTX2** v rozsahu 0 až 255 kΩ.

Nejdříve nastavíme registr **SPL** na konec datové paměti, abychom mohli používat zásobník (jak jsme již ukázali dříve). Vývod **PB2** (ovládá tranzistor T1) musí být konfigurován jako výstupní (**SBI DDRB,2**).

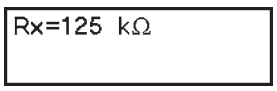
Následně inicializujeme LCD displej rutinou **INIT** a vypíšeme text **Rx= kΩ** (za rovnítkem jsou 4 znaky vynechány, slouží pro zobrazení mezery a tří číslic výsledku) - viz obr. 9.8.

Znaky **R**, **x** a **=** zobrazíme normálně. Před zobrazením dalších znaků musíme posunout kurzor na pozici 1, 8. Znak Ω je definován v tab. 8.2 (viz KE 5/2006) pod pořadovým číslem 244.

Vlastní měření začíná od návěští **HLAVNI**. Začneme vynulováním obsahu čítače (postupným zápisem 0 do registrů **TCNT1H**, **TCNT1L**) a vynulováním příznaku přetečení (zápisem 1 do bitu **TOV1**). Potom sepneme tranzistor T1 (**SBI PORTB,2**). Po ustálení (zajistí volání rutiny **CEKEJ**), tranzistor rozpojíme (**CBI PORTB,2**) a spustíme čítání impulsů zápisem hodnoty **START** (0b00000010) do registru **TCCR1B**. Tak bude čítán kmitočet odpovídající 1/8 hodinového kmitočtu procesoru. S ohledem na tento kmitočet byla provedena volba hodnot **RN1**, **RN2**, **CF** a rozsah **RX = 0 až 255 kΩ**.



Obr. 9.8. Úvodní výpis na LCD



Obr. 9.9. Stav displeje LCD při měření odporu 125 kΩ

*Tab 9.2. Program **PROG_18.ASM***

```
PROG_18.ASM:
        .NOLIST
        .INCLUDE "tn2313def.inc"
        .LIST

        .EQU START=0b00000010 ;spuštení cítace
        .EQU STOP =0b00000000 ;zastavení cítace

RESET:  LDI R16,RAMEND    ;nastav
        OUT SPL,R16       ;SPL
        SBI DDRB,2        ;PB2 je výstup
        RCALL INIT        ;inicializace LCD
        LDI A,'R'
        RCALL WRDATA      ;vypiš R
        LDI A,'x'
        RCALL WRDATA      ;vypiš x
        LDI A,'='
        RCALL WRDATA      ;vypiš =
        LDI A,1
        LDI B,8
        RCALL SETXY       ;kurzor na 1,8
        LDI A,'k'
        RCALL WRDATA      ;vypiš k
        LDI A,244
        RCALL WRDATA      ;vypiš Ω

HLAVNI: CLR R16           ;nuluj
        OUT TCNT1H,R16    ;TCNT1H
        OUT TCNT1L,R16    ;TCNT1L
        LDI R16,1<<TOV1   ;nuluj
        OUT TIFR,R16      ;TOV1
        SBI PORTB,2       ;vybij CN
        RCALL CEKEJ       ;ustálení
        CBI PORTB,2       ;konec vybíjení
        LDI R16,START     ;priprava na start
        OUT TCCR1B,R16    ;START
        LDI R16,STOP      ;príprava na STOP
KOMP:   SBIS ACSR,ACO     ;test ACO
        RJMP KOMP         ;testuj znovu
        ;ACO=1, konec merení:
        OUT TCCR1B,R16    ;STOP
        IN R16,TIFR       ;cti TIFR
        SBRC R16,TOV1     ;test TOV1
        RJMP PRET         ;preteklo
        ;zobrazení výsledku:
        IN R16,TCNT1L     ;cti obsah
        IN DD8U,TCNT1H    ;cítace
        LDI DV8U,10       ;delitel 10
        RCALL DIV8U       ;vypocítej podíl
        PUSH DREM8U       ;ulož jednotky
        LDI DV8U,10
        RCALL DIV8U       ;vypocítej podíl
        PUSH DREM8U       ;ulož desítky
        LDI DV8U,10
        RCALL DIV8U       ;vypocítej podíl

        LDI A,1
        LDI B,4
        RCALL SETXY       ;kurzor na 1,4
        MOV A,DREM8U      ;nahraj stovky
        SUBI A,-'0'
        RCALL WRDATA      ;zobraz stovky
        POP A
        SUBI A,-'0'
        RCALL WRDATA      ;zobraz desítky
        POP A
        SUBI A,-'0'
        RCALL WRDATA      ;zobraz jednotky
        RJMP HLAVNI

        ;obsluha pretecení:
PRET:   LDI A,1
        LDI B,4
        RCALL SETXY       ;kurzor na 1,4
        LDI A,'?'
        RCALL WRDATA      ;vypiš ?
        LDI A,'?'
        RCALL WRDATA      ;vypiš ?
        LDI A,'?'
        RCALL WRDATA      ;vypiš ?
        RJMP HLAVNI

        ;cekací rutina (ceká asi 0,5 s):
CEKEJ:  LDI R17,40
        LDI R18,0
        LDI R19,0
CEKEJA: DEC R19
        BRNE CEKEJA       ;smycka 1
        DEC R18
        BRNE CEKEJA       ;smycka 2
        DEC R17
        BRNE CEKEJA       ;smycka 3
        RET               ;návrat

        ;vložení rutin LCD:
        .INCLUDE "LCD.INC"
        ;vložení delicí rutiny:
        .INCLUDE "AVR200.INC"
```

Následně se do registru R16 nahraje hodnota odpovídající zastavení čítače (STOP = 0b00000000) a program přechází na návěští **KOMP**, které testuje příznak **ACO** analogového komparátoru. Je-li ACO = 0, pokračuje se v nabíjení. Pro ACO = 1 platí, že *U*+ > *U*-. V tom případě měření končí, zastavíme čítač.

Pokud je měřený odpor příliš velký, může nastat přetečení čítače. Nejdříve se tedy testuje příznak **TOV1**. Pokud je nastaven, přejde program na návěští **PRET**, které zajistí zobrazení textu **Rx=??? kΩ**.

Pokud čítač nepřetekl, zobrazíme naměřenou hodnotu. Pro převod do desítkové soustavy je použita rutina **DIV8U** (rutiny pro dělení a násobení lze stáhnout z WWW stránek společnosti ATMEL), která se do zdrojového souboru přidává ve formě souboru **AVR200.INC**.

Rutina **DIV8U** používá tyto registry: **DD8U** (R16) - dělenec, **DV8U** (R17) - dělitel, **DREM8U** (R15) - zbytek po dělení, **DRES8U** (R16) - podíl a jako pomocný registr používá **R18**. Při dělení deseti tedy nahrajeme dělenou hodnotu do **DD8U** a konstantu **10** zase do registru **DV8U**. Podíl je uložen do registru **DRES8U** a zbytek do registru **DREM8U**.

Z výsledku měření uvažujeme pouze horní bajt, tedy registr **TCNT1H**. Nejdříve však musíme přečíst registr **TCNT1L** (hodnotu zahodíme).

Převod do desítkové soustavy probíhá postupným dělením převáděné hodnoty. Zbytek po prvním dělení deseti představuje jednotky. Zbytek po druhém dělení jsou desítky a zbytek po třetím dělení jsou stovky.

Vzhledem k tomu, že nejdříve musíme zobrazit stovky, pak desítky a nakonec jednotky, je nutné zbytky po dělení ukládat instrukcí **PUSH DREM8U** do zásobníku. Při vybírání údajů nám zásobník poskytne přesně obrácené pořadí.

Před výpisem ještě musíme nastavit pozici kurzoru na 1, 4 (pozice za rovnítkem). Každý převedený zbytek zvýšíme instrukcí **SUBI R16,–'0'** o hodnotu znaku **'0'** a tak jej před zobrazením převedeme na číslici.

Přípravek ocejchujete tak, že na svorky **RX** připojíte rezistor o známém odporu, např. 100 kΩ. Změnou odporu rezistoru RN1 nebo RN2 dosáhneme zobrazení údaje 100 na displeji.

Program **PROG_18.ASM** je vypsán v tab. 9.2. Příklad najdete v [14] v adresáři **PROGRAMY\PROG_18**.

10. Jednotka USART

Univerzální synchronní a asynchronní sériový přijímač a vysílač (USART) je vysoce flexibilní zařízení pro sériovou komunikaci.

Jeho klíčové vlastnosti jsou:

- plný duplex (může současně přijímat a vysílat),
- synchronní nebo asynchronní režim,
- při synchronním režimu může pracovat jako Master (nadřízený obvod) nebo Slave (podřízený obvod),
- generátor rychlosti má velké rozlišení (snadné nastavení žádané přenosové rychlosti),
- podpora rámců délky 5 až 9 datových bitů a 1 až 2 stop-bity,
- generátor sudé/liché parity pro vysílač a hardwarové testování parity pro přijímač,
- detekce ztráty znaku (OverRun),
- detekce chyby rámce (Frame Error),
- obsahuje digitální dolnofrekvenční propust pro detekci falešného start-bitu a filtraci zákmitů datových bitů,
- tři nezávislá přerušení (odvysílání znaku, vyprázdnění vysílacího registru a příjem znaku),
- víceprocesorový komunikační režim,
- možnost zdvojnásobit přenosovou rychlost v asynchronním režimu.

10.1. Základní vlastnosti

Na obr. 10.1 je zjednodušené blokové schéma jednotky USART včetně klíčových registrů. Hlavními bloky jsou:

- **Generátor hodin**, který obsahuje synchronizační logiku pro vnější hodinový vstup používaný synchronním sériovým režimem.
- **Vysílač**, který obsahuje jeden zapisovací buffer, sériový posuvný registr, generátor parity a řídicí jednotku pro obsluhu různých sériových rámců. Zapisovací buffer zajišťuje kontinuální přenos dat bez jakéhokoli zpoždění mezi rámci.
- **Přijímač**, který je nejkomplikovanější částí jednotky USART, protože obsahuje jednotky pro obnovu (rekonstrukci) hodin a dat. Tyto jednotky jsou použity pro příjem asynchronních dat. Přijímač dále obsahuje detektor parity, řídicí logiku, posuvný registr a dvouúrovňový přijímací buffer (UDR). Přijímač podporuje stejné formáty rámců jako vysílač a dále zajišťuje detekci chyby rámce, ztráty znaku a chybu parity.

Kompatibilita mezi USART a UART

Jednotka USART zakomponovaná do mikrořadičů ATtiny2313 je plně kompatibilní s jednotkami UART, které byly k dispozici ve starších modelech mikrořadičů AVR. Je zachováno:

- umístění bitů uvnitř všech USART registrů,
- generátor přenosové rychlosti,
- funkce vysílače,
- funkčnost vysílacího bufferu,
- funkce přijímače.

Bufferování příjmu má dvě vylepšení, která mohou mít vliv na kompatibilitu v některých speciálních případech:

- Přijímač má dvouúrovňový buffer. Tento buffer pracuje jako kruhová fronta (po přečtení prvního údaje je k dispozici druhý v pořadí). Proto se z registru **UDR** musí přijatý údaj číst pouze jednou! Více důležitý je fakt, že příznaky chyby (**FE** a **DOR**) a devátý přijatý bit (**RXB8**) jsou bufferovány s daty v přijímacím bufferu. Proto se musí stavové bity číst vždy před čtením registru **UDR**. Jinak se ztratí informace o chybě.
- Posuvný registr přijímače může pracovat dokonce jako tříúrovňový buffer. To je zajištěno udržováním přijatých dat v sériovém posuvném registru v případě, že je dvouúrovňový buffer plný až do detekce nového start-bitu. USART je tedy nyní více odolný ztrátě znaku.
- Byly přejmenovány některé bity (ovšem funkce byla zachována). Bit **CHR9** byl přejmenován na **UCSZ2**, bit **OR** byl přejmenován na **DOR**.

10.2. Generátor hodin

Generátor hodin vytváří bázové hodiny pro vysílač a přijímač. Jednotka USART podporuje čtyři režimy práce hodin:

- normální asynchronní režim,
- asynchronní režim s dvojnásobnou rychlostí,
- synchronní režim master,
- synchronní režim slave.

Mezi synchronním a asynchronním režimem vybírá bit **UMSEL** z registru **UCSRC**. Dvojnásobnou rychlost (pouze v asynchronním režimu) volí bit **U2X** z registru **UCSRA**.

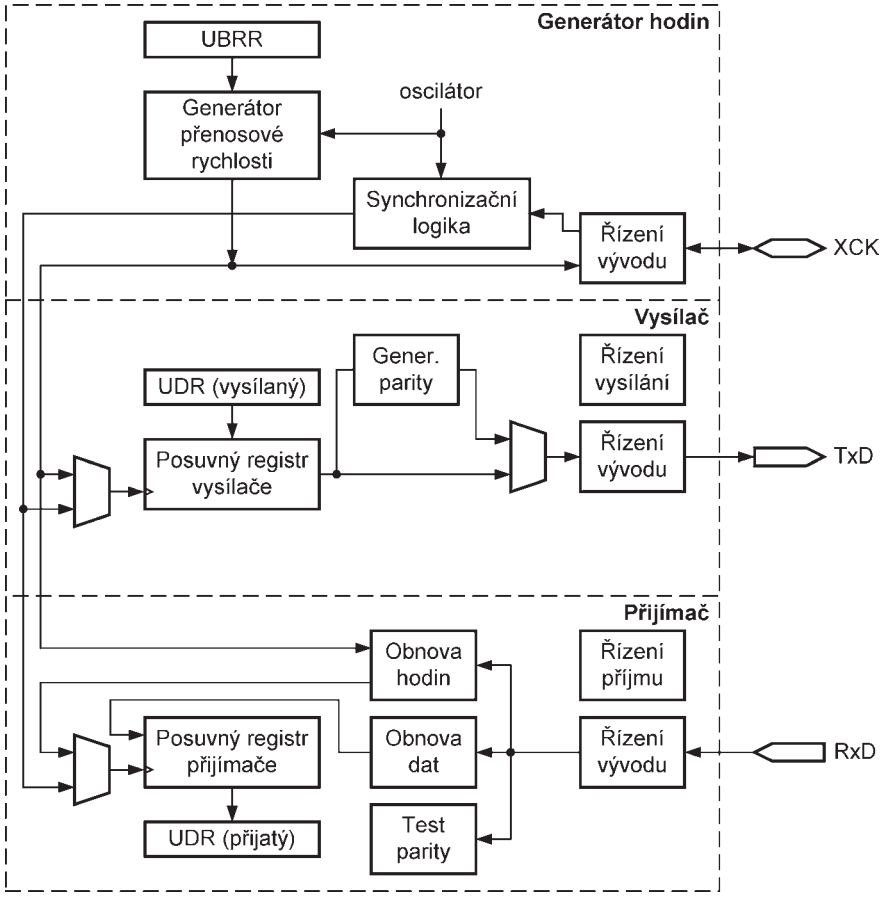
Při použití synchronního režimu (UMSEL = 1) určuje bit **DDRD2** (druhý bit registru DDRD), zda je hodinový signál na vývodu **XCK** (PD2) vnitřní (master) nebo vnější (slave). Vývod XCK je aktivní pouze v synchronním režimu.

Vnitřní generátor hodin - generátor přenosové rychlosti

Vnitřní generátor hodin se používá pro asynchronní a synchronní master režimy.

16bitový registr **UBRR** (UBRRH, UBRRL) a dolů čítající čítač zajišťují funkci programovatelné předděličky, tedy vlastně generátoru přenosové rychlosti. Výsledný kmitočet f_{BR} je dán vzorcem:

$$f_{BR} = f_0/(\text{UBRR} + 1),$$



Obr. 10.1. Blokové schéma jednotky USART

Tab 10.1. Vzorce pro výpočet přenosové rychlosti a hodnoty UBRR

Režim	Vzorec pro výpočet přenosové rychlosti	Vzorec pro výpočet hodnoty UBRR
Normální asynchronní režim (U2X = 0)	$f_{BR} = f_0/16\cdot(UBRR + 1)$	$UBRR = [f_0/(16\cdot f_{BR})] - 1$
Asynchronní režim s dvojnásobnou rychlostí (U2X = 1)	$f_{BR} = f_0/8\cdot(UBRR + 1)$	$UBRR = [f_0/(8\cdot f_{BR})] - 1$
Synchronní režim master	$f_{BR} = f_0/2\cdot(UBRR + 1)$	$UBRR = [f_0/(2\cdot f_{BR})] - 1$

kde f_0 je hodinový signál mikrořadiče.

Tyto hodiny jsou však dále děleny číslem 2, 8 nebo 16 podle zvoleného režimu - viz tab. 10.1.

Režim dvojnásobné rychlosti

Přenosovou rychlost v asynchronním režimu lze zdvojnásobit nastavením bitu **U2X** z registru **UCSRA**. Pro synchronní režim nemá tento bit význam (raději **U2X** vynulujte).

Nastavení bitu **U2X** vede ke snížení dělicího poměru z 16 na 8, to fakticky vede ke zdvojnásobení přenosové rychlosti. Poznamenejme, že přijímač pak vyhodnocuje pouze poloviční počet vzorků, takže je vyžadováno nastavit přenosovou rychlost přesněji.

Vnější hodiny

Vnější hodiny jsou používány v synchronním režimu slave.

Vnější hodinový vstup z vývodu **XCK** je vzorkován synchronizačním registrem. Tak se minimalizuje možná metastabilita. Výstup ze synchronizačního registru pak prochází detektorem hrany a teprve potom je použit přijímačem nebo vysílačem.

Tento proces vede ke zpoždění, které odpovídá dvěma periodám hodin mikrořadiče. Proto je kmitočet vnějších hodin dán nerovnicí:

$$f_{XCK} < f_0/4.$$

Synchronní funkce hodin

V synchronním režimu je vývod **XCK** použit buď jako vstup hodin (slave) nebo výstup hodin (master).

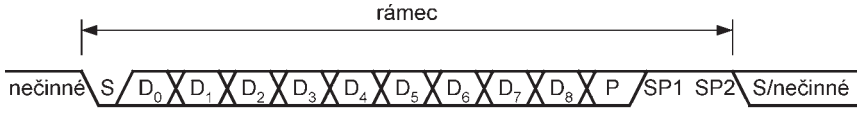
Základním principem je skutečnost, že vstup (vývod **RxD**) je vzorkován opačnou hranou hodin **XCK**, než kterou se mění výstup (vývod **TxD**).

Bit **UCPOL** z registru **UCRSC** vybírá hranu hodin **XCK** použitou pro vzorkování/změnu dat (viz tab. 10.2).

10.3. Formát rámce

Sériový rámec je definován jedním znakem datových bitů spolu se synchronizačními bity (start-bit a stop-bity) a volitelnou paritou pro test chyby. Jednotka USART podporuje všech 30 kombinací těchto parametrů:

- jeden start-bit,
- 5 až 9 datových bitů,
- žádná parita nebo jeden paritní bit (sudá nebo lichá parita),
- 1 nebo 2 stop-bity.



S – start-bit, log. 0

D_0 až D_8 – datové bity (5 až 9)

P – paritní bit (sudá nebo lichá parita)

SPn – stop-bity

Obr. 10.2. Formát rámce

Rámec začíná start-bitem, který je následován nejméně významným bitem. Následují další datové bity do maximálního možného počtu 9. Je-li povolena parita, je připojen paritní bit a rámec končí stop-bity. Formát rámce ilustruje obr. 10.2.

Když je rámec kompletně odeslán, může být přímo následován dalším rámcem. Komunikační linka také může zůstat v neaktivním stavu („log. 1“).

Formát rámce použitý jednotou USART je nastaven bity **USCZ2 až UCSZ0**, **UPM1 až UPM0** a **USBS** z registrů **UCSRB** a **UCSRC**. Přijímač i vysílač používají stejné nastavení. Změna nastavení libovolného z těchto bitů vede k chybě probíhajícího přenosu:

- Bity nastavující délku znaku (**UCSZ2 až UCSZ0**) vybírají počet datových bitů v rámci.
- Bity **UPM1 až UPM0** povolují a nastavují typ paritního bitu.
- Výběr jednoho nebo dvou stop-bitů je proveden bitem **USBS**. Přijímač však druhý stop-bit ignoruje. Chyba rámce bude detekována v případě, že první stop-bit má hodnotu „log. 0“.

Výpočet paritního bitu

Paritní bit je vypočítáván jako výlučný logický součet (EXOR, značí se ⊕) všech datových bitů. Pro lichou paritu je tento výsledek invertován.

Nebo to lze říci jinak: Celkový počet jedniček v datových bitech včetně paritního bitu je pro sudou paritu sudý a pro lichou paritu lichý:

$$P_{lichá} = D_{n-1} \oplus ... \oplus D_3 \oplus D_2 \oplus D_1 \oplus D_0 \oplus 0$$

$$P_{sudá} = D_{n-1} \oplus ... \oplus D_3 \oplus D_2 \oplus D_1 \oplus D_0 \oplus 1$$

10.4. Inicializace jednotky USART

Před zahájením komunikace musí být jednotka USART zinicializována.

Tento proces obvykle sestává z nastavení přenosové rychlosti, formátu rámce a povolení funkce přijímače a vysílače (podle požadovaného použití). Pro přerušením řízenou jednotku USART musí být při inicializaci zakázáno přerušení (I = 0).

Před provedením reinicializace (např. v důsledku změny přenosové rychlosti nebo formátu rámce) musíme zajistit dokončení dříve spuštěného přenosu. Příznak **TXC** lze použít pro test dokončeného vysílání. Příznak **RXC** lze použít pro test, že v přijímacím bufferu nejsou dosud nepřečtené znaky. Poznamenejme, že příznak **TXC** musí být vynulován před každým přenosem (tedy zápisem do registru **UDR**).

Následující jednoduchý příklad ukazuje inicializaci jednotky USART. Je zvolen asynchronní režim bez přerušení (obsluha událostí je řešena programově - polling). Formát rámce je stanoven pevně, přenosová rychlost je předávána jako parametr tohoto podprogramu (v registrech **R17, R16**).

```
USART_Init:     ;Nastav prenosovou rychlost:
                OUT UBRRH,R17
                OUT UBRRL,R16

                ;Povol prijímac i vysílac:
                LDI R16,(1<<RXEN)|(1<<TXEN)
                OUT UCSRB,R16

                ;Nastav formát rámce:
                ;8 datových bitů, 2 stop-bity
                LDI R16,(1<<USBS)|(3<<UCSZ0)
                OUT UCSRC,R16

                RET
```

Pozn.: *Symbol* **<<** *odpovídá operátoru bitového posunu doleva. Hodnota zapsaná vlevo se posune o tolik bitů,*

*Tab 10.2. Úloha bitu **UCPOL***

UCPOL	Změna dat na TxD	Vzorkování RxD
0	vzestupná hrana XCK	sestupná hrana XCK
1	sestupná hrana XCK	vzestupná hrana XCK

*kolik je uvedeno vpravo. Např. zápis **1<<4** způsobí přesunutí jednotkového bitu z nejnižšího řádu do řádu $2^4 = 16$. Zápis **3<<2** ($3_{10} = 00000011_2$) dá výslednou hodnotu 12 ($12_{10} = 00001100_2$).*

*Symbol **|** odpovídá operátoru bitového součtu (or, nebo). Je-li RXEN = 4 a TXEN=3, vede výraz **(1<<RXEN)|(1<<TXEN)** na výsledek 00011000_2. Tedy na nastavení 4. bitu a 3. bitu, které odpovídají povolení příjmu a vysílání. Závorky jsou nutné s ohledem na prioritu operátorů.*

10.5. Vysílání dat - USART vysílač

USART vysílač je povolen nastavením bitu **TXEN** z registru **UCSRB**.

Když je vysílač povolen, je normální funkce vývodu **TxD** (PD1) překryta do podoby sériového výstupu. Pro povolení výstupní funkce musí být bit **DDRD1** nastaven.

Přenosová rychlost, režim práce a formát rámce musí být nastaven dříve, než proběhne první přenos. Je-li použit synchronní režim, je vývod **XCK** (PD2) určen pro přenos hodinového signálu.

Vysílání rámců s 5 až 8 datovými bity

Přenos dat je zahájen vložením vysílaných dat do vysílacího bufferu. Tato činnost proběhne zápisem do registru **UDR**.

Bufferovaná data ve vysílacím bufferu budou přesunuta do posuvného registru vysílače v okamžiku, když je registr připraven vyslat nový rámec.

Do posuvného registru jsou nahrána nová data v okamžiku, když se nachází v nečinném stavu (neběží přenos) nebo okamžitě po odeslání posledního stop-bitu předchozího rámce.

Když je posuvný registr nahrán novými daty, bude vysílat jeden kompletní rámec rychlostí danou registrem **UBRR**, bitem **U2X** nebo hodinami na vývodu **XCK** podle toho, jaký je zvolen pracovní režim.

Následující příklad ukazuje USART vysílání založené na pollingu (bez použití přerušení) pomocí testování příznaku **UDRE**. Při použití rámců kratších než 8 bitů jsou vyšší bity zapsané do registru **UDR** ignorovány. Připomeňme, že jednotka USART musí být před vysíláním zinicializována:

```
USART_Transmit:   ;Cekej na vyprázdnení vysílacího
                  bufferu:
                  SBIS UCSRA,UDRE
                  RJMP USART_Transmit

                  ;Vlož data (R16) do bufferu, tím
                  se pošlou:
                  OUT UDR,R16

                  RET
```

Vysílání rámců s 9 datovými bity

Pro 9bitové znaky (UCSZ = 7) se devátý bit zapisuje do bitu **TXB8** z registru **UCSRB** před tím, než se nižší bity zapíšou do registru **UDR**.

Následující příklad ukazuje vysílání 9bitového znaku (devátý bit je uložen v nejnižším bitu registru R17, nižších osm bitů je uloženo v registru R16):

```
USART_Transmit:   ;Cekej na vyprázdnení vysílacího
                  bufferu:
                  SBIS UCSRA,UDRE
                  RJMP USART_Transmit

                  ;Kopie nejnižšího bitu R17 do
                  TXB8:
                  CBI UCSRB,TXB8
                  SBRC R17,0
                  SBI UCSRB,TXB8

                  ;Vlož data (R16) do bufferu, tím
                  se pošlou:
                  OUT UDR,R16

                  RET
```

Devátý bit lze také použít pro indikaci adresovacího rámce při použití víceprocesorové komunikace nebo například pro nějakou synchronizaci.

Příznaky a přerušení při vysílání

USART vysílač má dva příznaky indikující jeho stav, oba příznaky lze použít pro generaci přerušení:

- **UDRE (USART Data Register Empty)** - datový registr vysílače je prázdný,
- **TXC (Transmit Complete)** - vysílání rámce dokončeno.

Příznak **UDRE** indikuje, že vysílací buffer je připraven přijmout nový znak. Tento bit je nastaven, když je vysílací buffer prázdný. Tento příznak je vynulován, když buffer obsahuje nějaká vysílaná data. Pro kompatibilitu s dalšími součástkami vždy vynulujte tento příznak, když zapisujete do registru **UCSRA**.

Je-li nastaven bit **UDRIE** z registru **UCSRB**, dojde po nastavení příznaku **UDRE** (při současně povoleném přerušení, I = 1) k vyvolání přerušení na adrese **UDREaddr**.

Příznak **UDRE** je vynulován po zápisu do registru **UDR**. Takže při použití přerušení musí obslužná rutina na adrese **UDREaddr** buď zapsat nová data do registru **UDR** (čímž se příznak **UDRE** vynuluje) nebo zakázat přerušení (vynulováním bitu **UDRIE**). Jinak po dokončení obslužné rutiny se přerušení aktivuje znovu!

Příznak **TXC** je nastaven, když byl aktivní rámec definovaný obsahem vysílacího posuvného registru celý odeslán ven a nová data nejsou aktuálně ve vysílacím bufferu.

Příznak **TXC** se automaticky vynuluje po vstupu do obslužné rutiny umístěné na adrese **UTXCaddr**. Také jej lze vynulovat zápisem **1** do příznaku **TXC**.

Je-li nastaven bit **TXCIE** z registru **UCSRB**, bude po nastavení příznaku **TXC** (při současně povoleném přerušení, I = 1) vykonána obslužná rutina přerušení na adrese **UTXCaddr**. Obsluha přerušení nemusí příznak **TXC** nulovat, bude to provedeno automaticky.

Generátor parity

Generátor parity vypočítává paritní bit ze sériového datového rámce. Je-li **UPM1 = 1** (povolena funkce paritního bitu), vloží vysílací logika paritní bit mezi poslední datový bit a první stop-bit rámce, který se vysílá.

Typ parity je určen bitem **UPM0**. Pro **UPM0 = 0** se jedná o sudou paritu, pro **UPM0 = 1** se jedná o lichou paritu. Viz tab. 10.4.

Odstavení vysílače

Vynulování bitu **TXEN** (TXEN = 0) způsobí odstavení vysílače. Tuto operaci musíme provést až po dokončení aktuálního přenosu. Po odstavení vysílače se funkce vývodu **TxD** vrací do normálního režimu.

10.6. Přijímání dat - USART přijímač

USART přijímač je povolen nastavením bitu **RXEN** z registru **UCSRB**.

Když je přijímač povolen, je normální funkce vývodu **RxD** (PD0) překryta do podoby sériového vstupu.

Přenosová rychlost, režim práce a formát rámce musí být nastaven před čtením libovolného znaku. Je-li použit synchronní režim, je vývod **XCK** (PD2) určen pro přenos hodinového signálu.

Příjem rámců s 5 až 8 datovými bity

Přijímač začne přijímat data, když detekuje platný start-bit. Každý následující bit je vzorkován zvolenou přenosovou rychlostí nebo **XCK** hodinami a vsouván do posuvného registru přijímače. Tento proces pokračuje až do příjmu prvního stop-bitu. Druhý stop-bit přijímač ignoruje.

Když je přijat první stop-bit, je kompletní sériový rámec obsažený v přijímacím posuvném registru přesunut do přijímacího bufferu. Přijímací buffer lze číst přes registr **UDR**.

Následující ukázka kódu uvádí jednoduchý podprogram zajišťující příjem znaku založený na pollingu (testování příznaku **RXC**). Když rámec používá méně než 8 bitů, jsou vyšší bity vynulovány. Jednotka USART musí být před příjmem znaku zinicializována:

```
USART_Receive:    ;Cekání na prijatá data:
                  SBIS UCSRA,RXC
                  RJMP USART_Receive

                  ;Ulož prijatá data do registru
                  R16:
                  IN R16,UDR

                  RET
```

Příjem rámce s 9 datovými bity

Při použití 9bitového údaje (UCSZ = = 7) musí být devátý bit čten z bitu **RXB8** z registru **UCSRB** před čtením nižších bitů z registru **UDR**. Toto pravi-

dlo totiž zajistí správné uložení příznaků **FE**, **DOR** a **PE**.

Čtení registru **UDR** totiž vede ke změně fáze přijímací kruhové fronty a proto se následně změní stavy bitů **TXB8**, **FE**, **DOR** a **PE**.

Následující ukázka kódu představuje podprogram pro příjem 9bitového znaku a stavových bitů (devátý bit je uložen v nejnižším bitu registru R17, nižších osm bitů je uloženo v registru R16):

```
USART_Receive:    ;Cekání na prijatá data:
                  SBIS UCSRA,RXC
                  RJMP USART_Receive

                  ;Ctení FE, DOR, PE a RXB8 a potom
                  data:
                  IN R18,UCSRA
                  IN R17,UCSRB
                  IN R16,UDR

                  ;Pri chybí vrátí -1:
                  ANDI R18,(1<<FE)|(1<<DOR)|(1<<PE)
                  BREQ USART_RecNoErr
                  LDI R17,HIGH(-1)
                  LDI R16,LOW(-1)

USART_RecNoErr:   ;Uložení 9. bitu do nejnižšího
                  bitu R17:
                  LSR R17
                  ANDI R17,0x01

                  RET
```

Příznak a přerušení dokončeného příjmu

USART přijímač má příznak indikující stav přijímače. Jedná se o příznak **RXC**, který indikuje, že v přijímacím bufferu jsou dosud nepřečtená data.

Tento příznak je tedy nastaven, pokud existují nepřečtená data v přijímacím bufferu. Tento příznak je vynulován, pokud je přijímací buffer prázdný.

Při odstavení přijímače (**RXEN = 0**) je obsah přijímacího bufferu „spláchnut" a následně bit **RXC** vynulován.

Příznak **RXC** se vynuluje čtením registru **UDR**.

Je-li nastaven bit **RXCIE** z registru **UCSRB**, způsobí nastavení příznaku **RXC** (při současně povoleném přerušení, I = 1) aktivaci přerušovací rutiny uložené na adrese **URXCaddr**. Je-li použit příjem řízený přerušením, musí obslužná rutina číst obsah registru **UDR** a tím vynulovat příznak **RXC**. Jinak je po ukončení obsluhy přerušení vyvoláno znovu.

Příznaky chyby příjmu

USART přijímač má tři příznaky chyby obsažené v registru **UCSRA**. Při zápisu do registru **UCSRA** musí být všechny příznaky chyby vynulovány. Žádný z příznaků chyby negeneruje přerušení:

- **FE (Frame Error)** - chyba rámce,
- **DOR (Data Overrun)** - ztráta znaku,
- **PE (Parity Error)** - chyba parity.

Připomeňme, že z důvodu bufferování příznaků chyby se registr **USCRA** musí číst před čtením registru **UDR**.

Příznak **FE** indikuje stav, kdy první stop-bit rámce byl uložen do přijímacího bufferu. **FE = 0**, pokud byl stop-bit korektně přijat. **FE = 1**, pokud stop-bit nebyl korektní.

Při zápisu do registru **UCSRA** tento bit vždy vynulujte.

Tento příznak lze použít pro detekci rozbité synchronizace, také detekuje přerušení linky. Na chování příznaku **FE** nemá vliv nastavení bitu **USBS** z registru **UCSRC**. Pro kompatibilitu s dalšími mikrořadiči je však doporučeno bit **USBS** vynulovat.

Příznak **DOR** indikuje ztrátu dat v důsledku přeplnění. Tato situace nastane, když je buffer přijímače plný (dva znaky), nový znak vyčkává v posuvném registru přijímače a je detekován nový start-bit.

Při zápisu do registru **UCSRA** tento bit vždy vynulujte. Příznak **DOR** se automaticky vynuluje po úspěšném přesunu dat z posuvném registru do bufferu přijímače.

Příznak **PE** indikuje, že přijatý rámec má chybu parity. Pokud není test parity povolen, je vždy **PE = 0**. Při zápisu do registru **UCSRA** tento bit vždy vynulujte.

Testování parity

Obvod testování parity je aktivní, když je nastaven bit **UPM1**. Typ parity se vybírá bitem **UPM0**.

Při povolení funkce se počítá parita datových bitů přijatého rámce a porovnává se s přijatým paritním bitem. Výsledek porovnávání je uložen v přijímacím bufferu společně s přijatými datovými bity. Příznak **PE** pak může být testován pro zjištění chyby parity.

Příznak **PE** zůstává platný do nového čtení registru **UDR**.

Odstavení přijímače

Na rozdíl od vysílače, je odstavení přijímače okamžité. Data z probíhajícího příjmu budou ztracena.

Po odstavení (**RXEN = 0**) přijímače se funkce vývodu **RxD** vrací do normálního režimu. Obsah bufferu je spláchnut, data se ztratí.

Spláchnutí bufferu přijímače

Buffer přijímače se spláchne po odstavení přijímače. Spláchnutí lze provést i programově - viz níže uvedený podprogram:

```
USART_Flush:      SBIS UCSRA,RXC
                  RET
                  IN R16, UDR
                  RJMP USART_Flush
```

10.7. Příjem asynchronních dat

Jednotka USART zahrnuje logiku pro rekonstrukci hodin a dat.

Logika rekonstrukce hodin je použita pro synchronizaci vnitřního generátoru přenosové rychlosti s příchozím sériovým rámcem na vývodu **RxD**.

Logika rekonstrukce dat vzorkuje a pomocí dolnofrekvenční propusti filtruje každý příchozí bit. Tím se zvyšuje šumová imunita přijímače. Rozsah asynchronního příjmu je závislý na přesnosti vnitřního generátoru přenosové rychlosti, rychlosti příchozích rámců a počtu bitů v rámci.

Rekonstrukce asynchronních hodin

Logika rekonstrukce hodin synchronizuje vnitřní hodiny s příchozím sériovým rámcem.

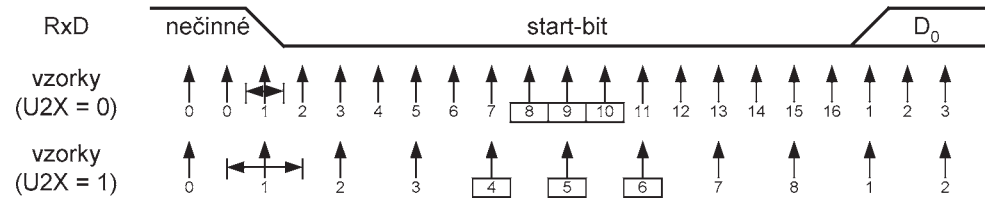
Na obr. 10.3 je vzorkovací proces start-bitu příchozího rámce. Vzorkovací rychlost je 16× vyšší než přenosová rychlost pro normální režim a 8× vyšší v režimu dvojnásobné rychlosti. Horizontální šipky naznačují variantu synchronizace pro normální režim (**U2X = 0**). Vzdálenější okamžiky vzorkování odpovídají dvojnásobné rychlosti (**U2X = 1**). Vzorky hodnoty 0 odpovídají okamžiku, kdy je linka **RxD** nečinná (bez komunikační aktivity). Když logika rekonstrukce hodin detekuje na lince **RxD** změnu z „log. 1" (nečinný) do „log. 0" (start-bit), vyvolá se detekční sekvence.

Představme si, že vzorek 1 odpovídá prvnímu vzorku nulové hodnoty. Logika rekonstrukce hodin používá vzorky 8, 9 a 10 v normálním režimu a vzorky 4, 5 a 6 v režimu dvojnásobné rychlosti pro rozhodnutí, zda byl přijat platný start-bit.

Pokud dva nebo tři z těchto vzorků mají hodnotu „log. 1", je start-bit posuzován jako šum a přijímač hledá další sestupnou hranu. Pokud je detekován platný start-bit, rekonstrukční logika je synchronizována a začíná rekonstrukce dat. Synchronizační proces se opakuje pro každý start-bit.

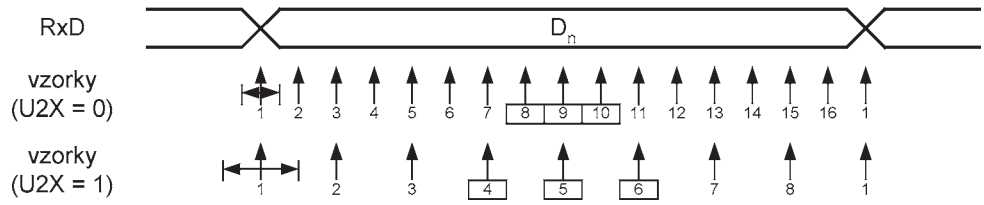
Rekonstrukce asynchronních dat

Když jsou hodiny přijímače synchronizovány se start-bitem, začíná rekonstrukce dat.

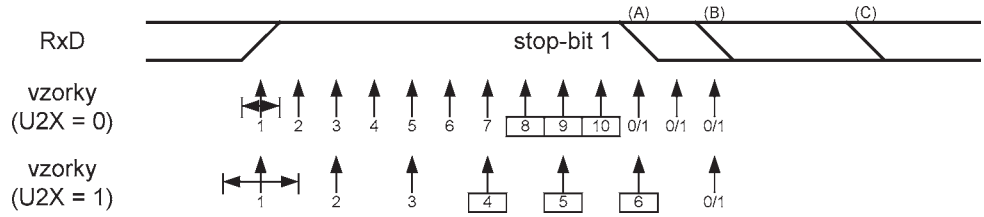


Obr. 10.3. Vzorkování start-bitu

Obr. 10.4. Vzorkování datových bitů a paritního bitu



Obr. 10.5. Vzorkování stop-bitu a následujícího start-bitu



Jednotka rekonstrukce dat používá stavový stroj, který má 16 fází pro každý bit v normálním režimu a 8 fází pro režim dvojnásobné rychlosti.

Na obr. 10.4 je vzorkování datových bitů a paritního bitu. Logika vzorkuje tři vzorky uprostřed přijatého bitu. Výsledná hodnota bitu je brána jako většina z těchto tří vzorků (výsledek je dán dvěma nebo třemi souhlasnými vzorky), tím je realizována dolnofrekvenční propust.

Proces rekonstrukce je opakován do kompletního příjmu rámce včetně stop-bitu. Poznamenejme, že přijímač vzorkuje pouze první stop-bit rámce.

Na obr. 10.5 je vzorkování stop-bitu a nejkratší možný začátek start-bitu dalšího rámce.

Pokud je stop-bit registrován jako „log. 0“, nastaví se příznak **FE**. Nová sestupná hrana indikuje start-bit nového rámce. Pro normální režim je první možný okamžik označen jako (A). Pro režim dvojnásobné rychlosti musí být první log. 0 zpožděna do okamžiku (B). Okamžik (C) určuje stop-bit plné délky.

Rozsah asynchronního režimu

Pracovní režim přijímače je závislý na rozdílech mezi rychlostí přijímaného signálu a vnitřního generátoru přenosové rychlosti.

Pokud vysílač posílá rámce příliš rychle nebo příliš pomalu nebo když vnitřně generovaná rychlost přijímače není podobná základnímu kmitočtu, není přijímač schopen synchronizovat rámce podle start-bitu.

Následující vzorce lze použít pro výpočet poměru rychlosti příchozího rámce a vnitřně generované přenosové rychlosti:

$$R_{pomalý} = [(D + 1)\cdot S]/(S - 1 + D\cdot S + S_F),$$

$$R_{rychlý} = [(D + 2)\cdot S]/[(D + 1)\cdot S + S_M],$$

kde:

- **D** je počet datových bitů včetně parity (D = 5 až 10),
- **S** je počet vzorků na bit (S = 16 pro normální režim, S = 8 pro režim dvojnásobné rychlosti),
- $\mathbf{S_F}$ je číslo prvního vzorku (S_F = 8 pro normální režim, S_F = 4 pro režim dvojnásobné rychlosti),
- $\mathbf{S_M}$ je číslo prostředního vzorku (S_M = 9 pro normální režim, S_M = 5 pro režim dvojnásobné rychlosti),
- $\mathbf{R_{pomalý}}$ je poměr nejpomalejší příchozí rychlosti, kterou lze akceptovat ku dané rychlosti přijímače,
- $\mathbf{R_{rychlý}}$ je poměr nejrychlejší příchozí rychlosti, kterou lze akceptovat ku dané rychlosti přijímače.

Tab 10.3. Doporučená maximální chyba přenosové rychlosti přijímače

D	Doporučená max. chyba [%] pro U2X = 0	Doporučená max. chyba [%] pro U2X = 1
5	±3,0	±2,5
6	±2,5	±2,0
7	±2,0	±1,5
8	±2,0	±1,5
9	±1,5	±1,5
10	±1,5	±1,0

Doporučená maximální chyba přenosové rychlosti přijímače je uvedena v tab. 10.3.

Jsou dva možné zdroje chyby přenosové rychlosti:

1. Existuje vždy minimální nestabilita generátoru hodin, která je způsobena kolísáním napájecího napětí. Může se jednat až o 2 %!

2. Druhou příčinu lze více ovlivnit. Jedná se o skutečnost, že přenosovou rychlost nelze vždy nastavit jako celočíselný poměr hodinového signálu mikrořadiče. Chybu lze zmenšit volbou **UBRR** poskytující nižší chybu.

10.8. Víceprocesorový komunikační režim

Nastavením bitu **MPCM** z registru **UCSRA** je povolena filtrační funkce příchozích rámců přijatých USART přijímačem.

Rámce, které neobsahují adresní informaci, budou ignorovány a nebudou ukládány do přijímacího bufferu. Tím je efektivně snížen počet příchozích rámců, které musí zpracovat procesor. Nastavení bitu MPCM nemá vliv na funkci vysílače.

Pokud je přijímač nastaven na příjem rámců obsahujících 5 až 8 bitů, pak první stop-bit indikuje, jestli rámec obsahuje data nebo adresu. Je-li přijímač nastaven na rámce šířky 9 datových bitů, je pro identifikaci rámce použit devátý bit (RxB8).

Je-li identifikační bit = 1 (tedy buď první stop-bit nebo devátý datový bit), obsahuje rámec adresu. Pokuď je identifikační bit = 0, obsahuje rámec data.

Víceprocesorový komunikační režim dovoluje, aby data z nadřízeného mikrořadiče přijímalo více podřízených mikrořadičů. Při přenosu se začne adresním rámcem. Je-li některý z mikrořadičů naadresován, přijímá další rámce normálně jako data. Současně další mikrořadiče přijaté rámce ignorují až do chvíle, když přijde další adresní rámec.

Použití bitu MPCM

Mikrořadič pracující jako master může používat 9bitový znak (UCSZ = 7). Devátý bit (**TXB8**) musí mít hodnotu 1 pro adresní rámec nebo 0 pro datový rámec. Podřízený mikrořadič musí v tomto případě používat také 9bitový formát znaku.

Dále uvedený postup ukazuje, jak lze uskutečnit výměnu dat ve víceprocesorovém komunikačním režimu:

1. Všechny podřízené mikrořadiče mají **MPCM = 1**.

2. Nadřízený mikrořadič vyšle adresní rámec a všechny podřízené mikrořadi-

če tento rámec čtou. V tomto případě dojde k nastavení příznaku **RXC** stejně, jako k němu dochází po příjmu znaku v normálním režimu.

3. Každý podřízený mikrořadič čte registr **UDR** a určuje, má-li být naadresován. Pokud ano, vynuluje si bit **MPCM**. V opačném případě ponechá bit **MPCM** nastaven a vyčkává na další adresní rámec.

4. Naadresovaný mikrořadič bude přijímat všechny datové rámce tak dlouho, dokud není přijat další adresní rámec. Ostatní mikrořadiče měly bit **MPCM** nastavený, takže datové rámce ignorovaly.

5. Když je naadresovaným mikrořadičem přijat poslední datový rámec, nastaví si bit **MPCM** a podobně jako ostatní vyčkává na příjem nového adresního rámce.

6. Proces se pak opakuje od bodu 2.

Použití formátu, který obsahuje 5 až 8 datových bitů je možné, ale krajně nepraktické. Především se musí zvolit dva stop-bity, neboť první stop-bit je používán pro indikaci typu rámce.

Nepoužívejte instrukce **SBI/CBI** pro nastavení/vynulování bitu **MPCM**! Bit **MPCM** totiž sdílí stejnou adresu s příznakem **TXC** a takto může dojít k jeho nechtěnému vynulování.

10.9. Popis registrů jednotky USART

Dále je uveden popis všech registrů používaných jednotkou USART.

UDR - v/v datový registr

Datový buffer přijímače a vysílače sdílí stejnou adresu označenou jako registr **UDR** (viz obr. 10.6). Zápis do **UDR** směřuje do bufferu vysílače. Čtení z **UDR** oslovuje buffer přijímače.

Pro případ 5, 6, 7bitového znaku jsou nejvyšší bity vysílačem ignorovány a přijímačem vynulovány.

Do bufferu vysílače lze zapisovat pouze tehdy, když je nastaven příznak **UDRE** z registru **UCSRA**. V případě **UDRE = 0** jsou zapisovaná data ignorována. Pokud je posuvný registr přijímače prázdný, přesunou se data po zápisu do bufferu vysílače do posuvného registru vysílače. Tím se data začnou vysouvat na vývodu **TxD**.

Přijímací buffer obsahuje dvouúrovňovou kruhovou frontu. Fronta mění svoji fázi po každém přístupu do přijímacího bufferu. Z tohoto důvodu nepoužívejte instrukce **SBI/CBI** nad registrem **UDR**. Buďte také opatrní při používání instrukcí **SBIC/SBIS**, protože i ty mění fázi fronty přijímače.

UCSRA - řídicí a stavový registr A

Registr **UCSRA** obsahuje především příznaky komunikace a chyby příjmu (viz obr. 10.7).

Význam jednotlivých bitů:

• **RXC (příjem dokončen)** - nastaví se, když jsou k dispozici nepřečtená data v přijímacím bufferu. Bit se vynuluje po vyprázdnění bufferu (neobsahuje žádná nepřečtená data). Příznak **RXC** lze použít pro generování přerušení.

• **TXC (vysílání dokončeno)** - nastaví se, když je aktuální rámec vyslán ven a žádná nová data nejsou aktuálně přítomna v bufferu vysílače (**UDR**). Příznak **TXC** lze použít pro generování přerušení.

• **UDRE (datový registr prázdný)** - indikuje, že vysílací buffer (**UDR**) je připraven přijmout nová data. Pro **UDRE = 1** je buffer prázdný a připraven pro zápis. Příznak **UDRE** lze použít pro generování přerušení. Příznak **UDRE** se po resetu automaticky nastaví, tím je indikováno, že je přijímač připraven.

• **FE (chyba rámce)** - nastaví se, pokud se nepřijal platný stop-bit (místo 1 byla 0). Je platný do přečtení **UDR**. Příznak **FE** se vynuluje, pokud byl stop-bit úspěšně přijat. Při zápisu do registru **UCSRA** musí být **FE = 0**.

• **DOR (ztráta dat)** - nastaví se, pokud je buffer přijímače plný (dva znaky), nový znak čeká v posuvném registru přijímače a je detekován nový start-bit. Stav příznaku **DOR** je platný do čtení **UDR**. Při zápisu do registru **UCSRA** musí být **DOR = 0**.

• **PE (chyba parity)** - nastaví se, pokud přijatá a nově vypočítaná parita vzájemně nesouhlasí. Stav příznaku **PE** je platný do čtení **UDR**. Při zápisu do registru **UCSRA** musí být **PE = 0**.

• **U2X (dvojnásobná přenosová rychlost)** - má význam pouze pro asynchronní režim (pro synchronní režim musí být **U2X = 0**). Pro **U2X = 1** se dělicí poměr snižuje z 16 na 8, takže přenosová rychlost je efektivně dvojnásobná.

• **MPCM (víceprocesorový komunikační režim)** - pro **MPCM = 1** jsou všechny příchozí rámce přijaté USART přijímačem, které neobsahují adresu, ignorovány. Funkce USART vysílače není tímto bitem ovlivněna.

UCSRB - řídicí a stavový registr B

Registr **UCSRB** obsahuje především bity povolující přerušení nebo činnost různých bloků jednotky USART (viz obr. 10.8).

Význam jednotlivých bitů:

• **RXCIE (povolení přerušení při příjmu)** - pro **RXCIE = 1** je povoleno přerušení při nastavení příznaku **RXC**. Při **RXCIE = 1** a **I = 1** a dokončení příjmu (**RXC** nastaven) se vyvolá rutina přerušení umístěná na **URXCaddr**.

• **TXCIE (povolení přerušení při dokončení vysílání)** - pro **TXCIE = 1** je povoleno přerušení při nastavení příznaku **TXC**. Při **TXCIE = 1** a **I = 1** a dokončení vysílání (**TXC** nastaven) se vyvolá rutina přerušení umístěná na **UTXCaddr**.

• **UDRIE (povolení přerušení při vyprázdnění UDR)** - pro **UDRIE = 1** je povoleno přerušení při nastavení příznaku **UDRE**. Při **UDRIE = 1** a **I = 1** a vyprázdnění **UDR** (**UDRE** nastaven) se vyvolá rutina přerušení umístěná na **UDREAddr**.

• **RXEN (povolení přijímače)** - pro **RXEN = 1** je povolena funkce USART přijímače. Přijímač překryje normální funkci vývodu **RxD**. Odstavení přijímače (**RXEN = 0**) způsobí spláchnutí bufferu přijímače a zneplatnění příznaků **FE**, **DOR**, **PE**.

• **TXEN (povolení vysílače)** - pro **TXEN = 1** je povolena funkce USART vysílače. Vysílač překryje normální funkci vývodu **TxD**. Ostavení vysílače (**TXEN = 0**) by nemělo nastat dříve, než doběhl aktivní přenos.

• **UCSZ2 (délka znaku)** - společně s bity **UCSZ1**, **UCSZ0** z registru **UCSRC** volí počet datových bitů (délka znaku) použitých v rámci přijímače/vysílače. Viz tab. 10.5.

• **RXB8 (D8 přijatých dat)** - devátý bit přijatého znaku, musí být přečten před čtením spodních bitů z **UDR**.

*Obr. 10.6. Registr **UDR***

Bit	7	6	5	4	3	2	1	0
UDR (čtení)	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
UDR (zápis)	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	0	0	0	0

*Obr. 10.7. Registr **UCSRA***

Bit	7	6	5	4	3	2	1	0
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Čtení/zápis	R	R/W	R	R	R	R	R/W	R/W
Výchozí hodnota	0	0	1	0	0	0	0	0

*Obr. 10.8. Registr **UCSRB***

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Výchozí hodnota	0	0	0	0	0	0	0	0

*Obr. 10.9. Registr **UCSRC***

Bit	7	6	5	4	3	2	1	0
	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Čtení/zápis	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	0	1	1	0

*Obr. 10.10. Registr **UBRRH, UBRRL***

Výchozí hodnota	0	0	0	0	0	0	0	0
Čtení/zápis	R	R	R	R	R/W	R/W	R/W	R/W
Bit	15	14	13	12	11	10	9	8
UBRRH	–	–	–	–	UBRR11	UBRR10	UBRR9	UBRR8
UBRRL	UBRR7	UBRR6	UBRR5	UBRR4	UBRR3	UBRR2	UBRR1	UBRR0
Bit	7	6	5	4	3	2	1	0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	0	0	0	0

• **TXB8 (D8 vysílaných dat)** - devátý bit vysílaného znaku, musí být zapsán před zápisem spodních bitů do **UDR**.

UCSRC - řídicí a stavový registr C

Registr **UCSRC** obsahuje především bity pro volbu režimu jednotky USART.

Význam jednotlivých bitů:

• **UMSEL (režim USART)** - pro **UMSEL = 0** je zvolen asynchronní režim, pro **UMSEL = 1** je zvolen synchronní režim.

• **UPM1, UPM0 (režim parity)** - tyto bity povolují a nastavují generaci parity a její testování. Je-li parita povolena, automaticky se generuje a vysílá paritní bit v každém rámci. Přijímač bude generovat hodnotu parity pro příchozí data a porovnávat je s nastavením **UPM0**. Při chybě se nastaví příznak **PE** z registru **UCSRA** (viz tab. 10.4).

• **USBS (výběr stop-bitu)** - volí počet stop-bitů vložených vysílačem. Pro **USBS = 0** jeden stop-bit, pro **USBS = 1** dva stop-bity. Přijímač toto nastavení ignoruje.

• **UCSZ1, UCSZ0 (délka znaku)** - v kombinaci s bitem **USCZ2** z registru **UCSRB** volí počet datových bitů rámce pro přijímač a vysílač (viz tab. 10.5).

• **UCPOL (polarita hodin)** - má význam pouze pro synchronní režim (pro asynchronní režim musí být **UCPOL = 0**). **UCPOL** volí vzájemný vztah mezi výstupem dat, vzorkováním vstupu a synchronními hodinami na vývodu **XCK** (viz tab. 10.2).

*Tab 10.4. Význam bitů **UPM1**, **UPM0***

UPM1	UPM0	Režim parity
0	0	odstavena
0	1	vyhrazeno (nepoužívat)
1	0	povolena, sudá parita
1	1	povoleno. lichá parita

*Tab 10.5. Význam bitů **UCSZ2** až **UCSZ0***

UCSZ2	UCSZ1	UCSZ0	Šířka znaku
0	0	0	5 bitů
0	0	1	6 bitů
0	1	0	7 bitů
0	1	1	8 bitů
1	0	0	vyhrazeno
1	0	1	vyhrazeno
1	1	0	vyhrazeno
1	1	1	9 bitů

UBRRH, UBRRL - registr přenosové rychlosti

Tento pár registrů slouží pro nastavení přenosové rychlosti.

Význam jednotlivých bitů:

• **UBRR11 až UBRR0 (12 bitů)** - dělicí poměr přenosové rychlosti - viz kapitolu 10.2.

10.10. Přípravek ATRS232+

Pro základní použití jednotky USART je potřebný přípravek **ATRS232+**, který převádí signály TTL logiky na úrovně standardu RS-232 a tak umožní připojit jednotku USART na sériový port běžného počítače.

Pro připojení sériového portu k osobnímu počítači byl již dříve navržen přípravek **ATRS232** (viz [1]). Vzhledem k tomu, že port PD obsahuje velké množství vývodů vestavěných periferií, bylo nutné přípravek upravit na novou variantu nazvanou **ATRS232+**.

Schéma přípravku **ATRS232+** je na obr. 10.11. Přípravek obsahuje především konektor **COM** pro připojení k sériovému portu počítače. Pomocí standardního převodníku **MAX232** (IO1) se napěťové úrovně typu RS-232 převádějí na úrovně TTL. Jedná se o známé zapojení, které není třeba dále komentovat.

Dále jsou na přípravku dva vzájemně propojené konektory **P1 a P2**. Toto propojení funguje jako „rozdvojka". Libovolný z obou konektorů použijeme na připojení na port PD mikrořadiče proto, abychom mohli pracovat se sériovým portem počítače (bity PD0 a PD1). Na druhý konektor pak můžeme připojit další přípravek, který potřebuje pracovat s některým z ostatních vývodů portu PD.

Na přípravku jsou ještě dvě svorkovnice **T0 a T1** pro přímé napojení na vývody PD4 (T0) a PD5 (OC0B/T1).

Propojovací kabel mezi sériovým portem počítače a konektorem COM je jednoduchý. Vždy se musí vzájemně propojit vývody 2 a 2, 3 a 3 a 5 a 5. Zapojení ostatních linek není podstatné.

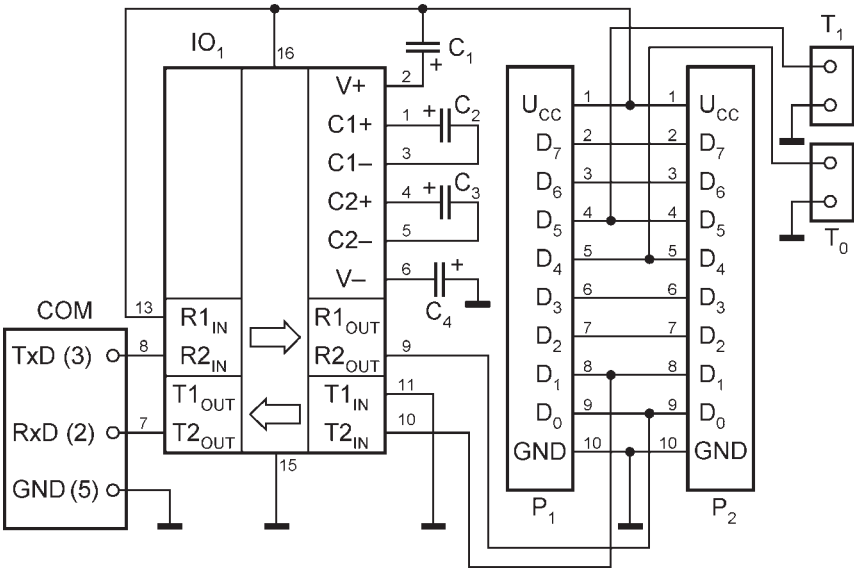
Výkresy potřebné pro zhotovení desky s plošnými spoji jsou na obr. 10.12 až obr. 10.14.

Seznam součástek pro ATRS232+

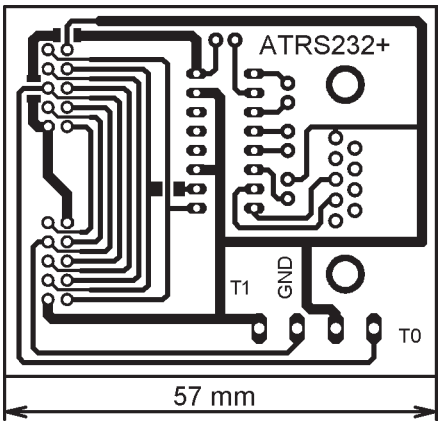
(cena asi 100 Kč):

R1 až R3	0 Ω, SMD 1206
C1 až C4	22 µF/16 V, radiální
IO1	ICL232 (MAX232)
COM	CAN 9 Z 90
P1, P2	MLW10G
T0, T1	ARK500/2

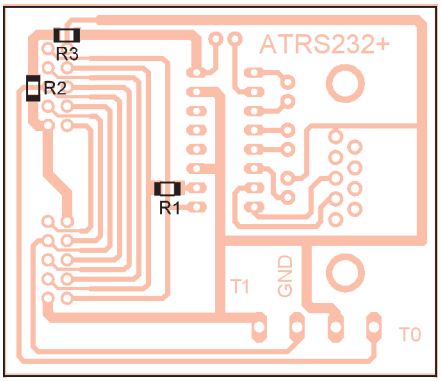
deska s plošnými spoji ATRS232+



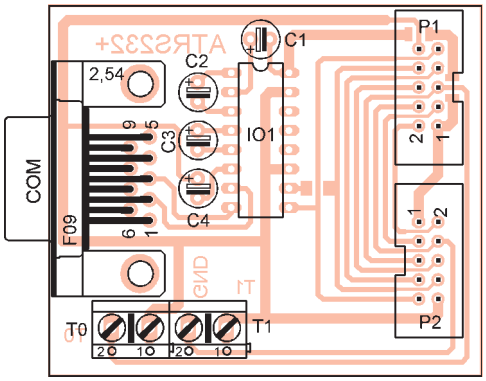
Obr. 10.11. Schéma přípravku ATRS232+



Obr. 10.12. Obrazec plošných spojů přípravku ATRS232+ (měř.: 1 : 1)



Obr. 10.13. Rozmístění součástek SMD na straně spojů na desce přípravku ATRS232+



Obr. 10.14. Rozmístění vývodových součástek na straně součástek na desce přípravku ATRS232+

10.11. Příklad 19 - ovládání přípravku AT8LED počítačem PC

Tento příklad již ukazuje základní komunikaci mezi počítačem PC a jednotkou USART. Jedná jak o příjem, tak i o vysílání.

Používá se jen jedno přerušení, protože každý přijatý znak je odeslán pro konrolu komunikace zpět.

Zadání: Připojte přípravek **ATRS232+** na port **PD**. Přípravek **AT8LED** připojte na port **PB**.

Přijměte jeden znak odeslaný z počítače PC a použijte jej na rozsvícení LED na přípravku **AT8LED**. Přijatý znak odešlete pro kontrolu komunikace zpět do PC.

Pro přenos použijeme asynchronní režim se šířkou znaku 8 bitů (jeden stop-bit, bez parity) a rychlostí 9600 Bd.

Výpočtem:

$$UBRR = [f_0/(16 \cdot f_{BR})] - 1 =$$

$$= [10000000/(16 \cdot 9600)] - 1 \approx 64$$

dojdeme k hodnotě **64**, která je zavedena jako symbol **BAUD** a následně nahrána do registrů **UBRRH**, **UBRRL**.

Pro příjem znaku je opět použito přerušení, takže na adrese **URXCaddr** najdeme skok na návěští **PRIJEM**, které danou obsluhu provede.

Inicializace nyní nastavuje registr **SPL** a konfiguruje port **PB** jako výstupní (sem se připojí přípravek **AT8LED**).

Důležité je konfigurovat vývod **PD1** jako výstupní, odpovídá lince **TxD**. Pokud to zapomeneme, nebude USART vysílač fungovat!

V registru **UCSRB** budou nastaveny bity: **RXEN** (povolení příjmu), **RXCIE** (povolení přerušení při příjmu), **TXEN** (povolení vysílání). Dále je třeba zvolit formát rámce (8 datových bitů + + 1 stop-bit) nastavením bitů **UCSZ0** a **UCSZ1** v registru **UCSRC**.

Obsluha přerušení **URXC** přečte přijatý znak z registru **UDR**, zneguje jej a předá do registru **PORTB** (tedy na přípravek **AT8LED**). Negace je nutná, připomeňme, že LED na přípravku **AT8LED** svítí při „log. 0“.

Potom se obsah registru **R16** zneguje zpátky a zápisem do registru **UDR** odešle zpět do počítače. Obsluha přerušení končí instrukcí **RETI**.

Celý tento program pro mikrořadič je nazván **PROG_19.ASM** a je vypsán v tab. 10.6. Příklad najdete v [14] v adresáři **PROGRAMY\PROG_19**.

Sestavení ovládacího programu pro počítač PC nebudeme komentovat, patří do knih jiné kategorie. Zájemcům můžeme doporučit [12] nebo [13].

Program pro počítač PC najdete ve stejném adresáři jako program pro mikrořadič (složka **WinApp**). Obsah adresáře musíte zkopírovat na pevný disk.

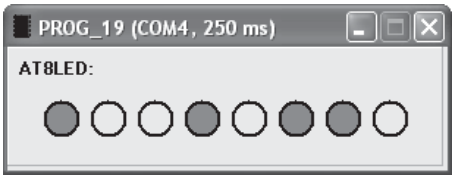
Velmi důležitý je soubor **PORT.INI**, který pomocí klíče **Port** určuje, na který port PC je mikrořadič připojen. Klíč **Interval** určuje interval (v ms) mezi odesláním dvou znaků z počítače PC:

```
PORT.INI:
[PORT]
Port=4

[TIMER]
Interval=250
```

V uvedeném výpisu se jednalo o port **COM4** (sériový port byl emulován převodníkem USB<=>RS-232), takže Port = 4. Interval byl nastaven na 250 ms (tedy 4× za sekundu).

*Tab 10.6. Program **PROG_19.ASM***

```
PROG_19.ASM:
        .NOLIST
        .INCLUDE "m16def.inc"
        .LIST

        .EQU BAUD=64        ;tj. 9600 Bd

        .CSEG
        RJMP RESET          ;inicializace
        .ORG URXCaddr
        RJMP PRIJEM         ;obsluha RXC

        ;inicializace:
RESET:  LDI R16,RAMEND      ;nastav
        OUT SPL,R16         ;SPL

        LDI R16,255         ;port B
        OUT DDRB,R16        ;je výstupní
        OUT PORTB,R16       ;zhasne LED

        SBI DDRD,1          ;TxD je výstup
        LDI R16,HIGH(BAUD) ;nastavení
        OUT UBRRH,R16       ;prenosové
        LDI R16,LOW(BAUD)  ;rychlosti
        OUT UBRRL,R16       ;do UBRR
        ;povol príjem+vysílání, a URXC prerušení:
        LDI R16,(1<<RXEN)|(1<<RXCIE)|(1<<TXEN)
        OUT UCSRB,R16
        LDI R16,3<<UCSZ0    ;8 datových bitů
        OUT UCSRC,R16       ;+1 stop-bit
        SEI                 ;I=0

        RJMP PC             ;cekej

        ;obsluha URXC:
PRIJEM: IN R16,UDR          ;nacti bajt
        COM R16             ;negace bitů
        OUT PORTB,R16       ;na LED
        COM R16             ;negace
        OUT UDR,R16         ;odešli zpet
        RETI                ;konec
```



*Obr. 10.15. Program **PROG_19.EXE** v akci*

Ověření funkce programu

Připojte přípravek **ATRS232+** pomocí konektoru COM k počítači PC. Potom spusťte ovládací program **PROG_19.EXE**. Na ploše monitoru PC se objeví okno s panelem **AT8LED** (obr. 10.15).

V panelu **AT8LED** klikáním na jednotlivá kolečka zadávejte hodnoty pro LED a sledujte je na přípravku **AT8LED**.

10.12. Příklad 20 - impulsní generátor ovládaný počítačem PC

Druhý příklad je již dosti komplikovaný. Pro příjem a vysílání bude používán buffer. Jak přijímání, tak vysílání bude používat přerušení. Navíc si ukážeme použití jednotky Output Compare čítače/časovače 0.

Zadání: Připojte přípravek **ATRS232+** na port **PD**.

Přijímejte z počítače PC dva bajty, které představují nastavení registrů

TCCR0B a **OCR0A**. Po příjmu obou bajtů odešlete pro kontrolu jejich hodnoty zpět do PC.

Pomocí registrů **TCCR0B** a **OCR0A** realizujte impulsní generátor s výstupem na vývodu **OC0B** (výstupní signál lze sledovat na svorkovnici **T1** přípravku **ATRS232+**).

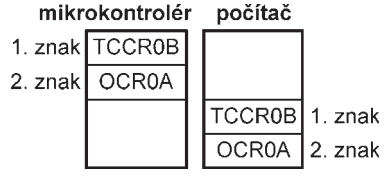
Pro přenos opět použijeme asynchronní režim se šířkou znaku 8 bitů (jeden stop-bit, bez parity) a rychlostí 9600 Bd. Výpočtem dojdeme opět k hodnotě 64, která je zavedena jako symbol **BAUD** a následně nahrána do registrů **UBRRH**, **UBRRL**.

Příjem a vysílání znaků je nyní realizováno přes buffer (vyrovnávací paměť). To dovoluje nejdříve přijmout oba znaky, které posílá počítač do mikrokontroléru a poté je za sebou odeslat (pro potvrzení příjmu). Buffer představuje 2 bajty, které začínají od návěští **BUFFER**. Registr **Z** je použit jako ukazovátko při práci s ním. Ve skutečnosti je horní bajt ukazatele **Z** vždy vynulován a mění se pouze obsah dolního bajtu (použit symbol **BUFPOS**). Buffer je záměrně umístěn na adresu, která odpovídá registrům **R19** až **R20** (zjednodušší to přenos přijatých znaků do řídicích registrů čítače/časovače 1). Při překladu nás na tuto skutečnost překladač upozorní (pro daný účel je to v pořádku).

Nyní je třeba obsluhovat dvě přerušení. Jedno při příjmu (adresa **URXCaddr**) označené jako **RX** a druhé při odvysílání (adresa **UTXCaddr**) označené jako **TX**.

Inicializace nastaví registr **SPL** a povolí výstup na vývodech **PD1** (TxD) a **PD5** (OC0B). Ukazatel **BUFPOS** se nastaví na počátek bufferu (na hodnotu **BUFFER**). Dále nastaví registry **UBRRL**, **UBRRH**, **UCSRC** a **UCSRB** (bity RXEN, RCIEN a TXCIE) a **TCCR0A** (vývod **OC0B** je v režimu toggle). Nakonec je povoleno přerušení a program se zacyklí instrukcí **RJMP PC**.

Příjem dat je zapsán od návěští **RX**. Nejdříve je uložen obsah stavového registru **SREG** do **IZASOB** (ne že by to bylo nutné, ale připomínáme vhodnost zachování obsahů registrů). Dále se čte obsah **UDR** a uloží se do buferu (adresa je dána **Z** resp. **BUFFER**). Následně se testuje, zda jsou již přijaty oba znaky (jednak proto, aby se buffer nepřeplnil a také proto, abychom mohli aktivovat vysílání). Jsou-li přijaty oba znaky, odpojí se příjem a aktivuje vysílání zápisem hodnoty **0b01011000** do **UCSRB**. Pro aktivaci vysílání je třeba zapsat první přijatý znak do **UDR** (zbylý znak se odvysílá pomocí obslužné rutiny



Obr. 10.16. Formát přenosu dat

TX) a s tím souvisí nastavení ukazatele **BUFPOS** na začátek buferu (**BUFFER**). Nakonec se obnoví stav **SREG**.

Vysílání dat je zapsáno od návěští **TX**. Přečte se znak z bufferu a odvysílá se zápisem do registru **UDR**. Pokud se odvysílají oba znaky, odpojí se vysílání a aktivuje příjem zápisem hodnoty **0b10011000** do **UCSRB**. Následně se dříve přijaté hodnoty (jsou uloženy v bufferu) nastaví do registrů **TCCR0B** (předdělička), **OCR0A** (požadovaný kmitočet). Tím, že je buffer umístěn do oblasti registrového pole, se tato operace zjednoduší. Jinak by se musela hodnota z bufferu přečíst do registru registrového pole a obsah tohoto registru zase zapsat do příslušného vstupně/výstupního registru. Takto se přijaté znaky ukládají přímo do registrů registrového pole, takže odpadá jeden přesun dat.

*Pozn.: Při realizaci tohoto příkladu se projevil jeden problém. Jedná se o to, že čítač/časovač 0 nemůže v žádném ze svých režimů používat jako vrchol obsah registru **OCR0B**. Pro tento účel lze používat pouze registr **OCR0A**. Na přípravku **ATRS232+** však není vývod **OC0A** k dispozici.*

*Proto byla použita drobná „finta" spočívající v tom, že se druhým přijatým bajtem nenastavuje obsah registru **OCR0B**, ale **OCR0A**. Aby byla jednotka **0C0B** schopna na vývodu **OC0B** generovat požadovaný signál, je registr **OCR0B** vždy nastaven na hodnotu 0. Takže stav daného vývodu se mění na začátku cyklu, ale synchronně s činností jednotky **OC0A**.*

Formát přenosu dat je zřejmý z obr. 10.16. Mikrořadič posílá všechny přijaté bajty zpět do počítače PC, tím se zjišťuje bezchybnost komunikace.

Celý tento program pro mikrořadič je nazván **PROG_20.ASM a** je vypsán v tab. 10.7. Příklad najdete v [14] v adresáři **PROGRAMY\PROG_20**.

Sestavení ovládacího programu pro počítač PC nebudeme opět komentovat. Program pro PC najdete ve stejném adresáři jako program pro mikrořadič (složka **WinApp**). Obsah tohoto adresáře musíte zkopírovat na pevný disk PC.

Velmi důležitý je soubor **PORT.INI**, který pomocí klíče **Port** určuje, na který port je mikrořadič připojen. Další klíče odpovídají poslednímu nastavení, které provedl uživatel:

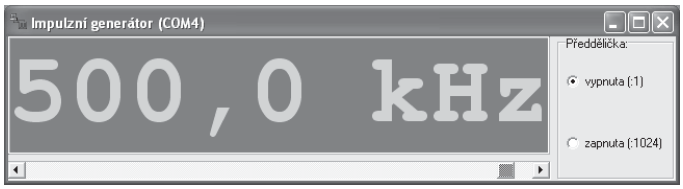
```
PORT.INI:
[PORT]
Port=4

[NASTAVENI]
Preddelicka=0
Delicka=-23
```

V uvedeném výpisu se jednalo o port **COM4** (sériový port byl emulován převodníkem USB<=>RS-232), takže Port = 4.

*Tab 10.7. Program **PROG_20.ASM***

```
PROG_20.ASM:
        .NOLIST
        .INCLUDE "tn2313.inc"
        .LIST

        .EQU BAUD=64        ;tj. 9600 Bd

        .DEF REG=R16        ;pracovní registr
        .DEF IREG=R17       ;registry pro
        .DEF IZASOB=R18     ;podporu prerušení

        .DSEG               ;datový segment
        .ORG 19             ;adresa 19=R19
BUFFER: .BYTE(2)            ;2bajtový buffer
                            ;v prostoru
                            ;registrů R19 až R20
        .EQU BUFKON=BUFFER+2;konec bufferu
        .DEF RTCCR0B=R19    ;1.bajt
        .DEF ROCR0B=R20     ;2.bajt
        .DEF BUFPOS=R30     ;ukazovátko v bufferu

        .CSEG               ;kódový segment
        RJMP RESET
        .ORG URXCaddr       ;USART RXC
        RJMP RX
        .ORG UTXCaddr       ;USART TXC
        RJMP TX

        ;inicializace
RESET:  LDI REG,RAMEND
        OUT SPL,REG         ;nastavení SPL

        SBI DDRD,1          ;TxD jako výstup
        SBI DDRD,5          ;OC0B jako výstup

        CLR ZH              ;BUFPOS(Z) ukazuje
        LDI BUFPOS,BUFFER   ;na zacátek bufferu

        LDI REG,LOW(BAUD)
        OUT UBRRL,REG       ;9600 Bd
        LDI REG,HIGH(BAUD)
        OUT UBRRH,REG
        ;8 datových bitů+1 stop-bit:
        LDI R16,3<<UCSZ0
        OUT UCSRC,R16
        LDI REG,0b10011000
        OUT UCSRB,REG       ;povol RXC
        LDI REG,0b00010010
        OUT TCCR0A,REG      ;toggle OC0B,CTC
        SEI                 ;I=1

        RJMP PC             ;smycka

        ;obsluha USART RXC:
RX:     IN IZASOB,SREG      ;ulož SREG
        IN IREG,UDR         ;cti data
        ST Z+,IREG          ;ulož do bufferu
        CPI BUFPOS,BUFKON+1 ;konec?
        BRNE RXKON          ;ne->skoc na RXKON
        ;prijato vše:
        LDI IREG,0b01011000
        OUT UCSRB,IREG      ;povol TXC
        LDI BUFPOS,BUFFER   ;BUFPOS na zacátek
        LD IREG,Z+          ;cti první bajt
        OUT UDR,IREG        ;a pošli jej
RXKON:  OUT SREG,IZASOB     ;obnov SREG
        RETI                ;konec obsluhy

        ;obsluha USART TXC:
TX:     IN IZASOB,SREG      ;ulož SREG
        LD IREG,Z+          ;cti bajt
        OUT UDR,IREG        ;a pošli jej
        CPI BUFPOS,BUFKON+1 ;konec?
        BRNE TXKON          ;ne->skoc na TXKON
        ;posláno vše:
        LDI IREG,0b10011000
        OUT UCSRB,IREG      ;povol RXC
        LDI BUFPOS,BUFFER   ;BUFPOS na zacátek
        OUT TCCR0B,RTCCR0B ;nastav TCCR0B,
        OUT OCR0A,ROCR0B    ;OCR0A
        LDI IREG,0          ;vynuluj
        OUT OCR0B,IREG      ;OCR0B
TXKON:  OUT SREG,IZASOB     ;obnov SREG
        RETI                ;konec obsluhy
```

Obr. 10.17. Program **PROG_20.EXE** v akci

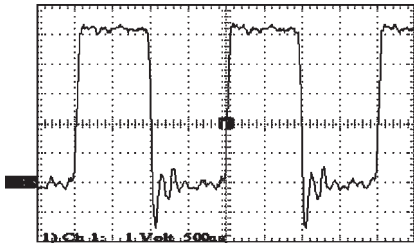


Ověření funkce programu

Připojte přípravek **ATRS232+** pomocí konektoru COM k počítači PC. Potom spusťte ovládací program **PROG_20.EXE**. Při běžícím programu je na monitoru PC okno podle obr. 10.17.

Výstupní signál lze sledovat na svorkovnici **T1** (obr. 10.18).

Kmitočet lze ladit jemně pomocí skrolbaru nebo hrubě tak, že požijeme buď přímo hodinový kmitočet krystalu, nebo jej nejdříve necháme vydělit 1024× (skupina Předdělička). Rozsah generovaných kmitočtů je **19,07 Hz až 5 MHz**.



Obr. 10.18. Výsledný průběh pro kmitočet 500 kHz (naprázdno)

10.13. Shrnutí

Nakonec si shrneme nejdůležitější poznatky, které byly podány v předchozím textu:

• **Hodnoty bitů lze zapisovat buď přímo v dvojkové soustavě nebo lze používat operátory << a |.** Pokud nás unavuje vyhledávat pozice bitů v jednotlivých registrech, můžeme použít operátor **<<**, který nám zadanou hodnotu (uvedenou vlevo) posune o žádaný počet bitů doleva (uvádí se vpravo). Při nastavování několika bitů současně můžeme použít operátor **|**. Prioritu řeší závorky. Např. výraz: **(1<<RXEN)|(1<<RXCIE)|(1<<TXEN)** odpovídá požadavku nastavení bitů **RXEN**, **RXCIE** a **TXEN**, tedy hodnotě **0b01011000**.

• **Pro příjem vyššího množství dat je dobré používat vyrovnávací paměť - buffer.** V našem příkladu byly přijímány dva bajty, příklad lze však snadno rozšířit na příjem vyššího množství dat. Při práci s bufferem používáme ukazatele a s nimi spojené instrukce. Např. instrukce **ST Z+,IREG** uloží na adresu obsaženou v Z obsah registru IREG a dále posune Z na následující adresu. Podobně instrukce **LD IREG,Z+** načte údaj z adresy, která je dána obsahem registru Z do registru IREG a následně posune Z na následující adresu.

11. Jednotka USI

Univerzální sériové rozhraní (USI) poskytuje základní hardwarové zdroje pro sériovou komunikaci. V kombinaci s minimem programového řízení zajišťuje významně vyšší přenosové rychlosti než řešení založené pouze na programovém řízení komunikace. Pro minimalizaci zátěže procesoru jsou k dispozici přerušení. Hlavní vlastnosti USI jsou:

• 2drátový synchronní přenos dat (master nebo slave, $f_{SCLMAX} = f_{CK}/16$),

• 3drátový synchronní přenos dat (master $f_{SCKMAX} = f_{CK}/16$, slave $f_{SCKMAX} = f_{CK}/4$),

• přerušení při příjmu dat,

• probuzení z režimu Idle,

• ve 2drátovém režimu je možné probuzení ze všech úsporných režimů včetně režimu Power-down,

• ve 3drátovém režimu je detektor startu vybaven přerušením.

11.1. Úvod

Na obr. 11.1 je zjednodušené blokové schéma jednotky USI.

8bitový posuvný registr (**USIDR**) je přímo přístupný přes datovou sběrnici a obsahuje příchozí a odchozí data. Registr není bufferovaný, takže data musí být přečtena tak rychle, jak je jen možné (jinak dojde k jejich ztrátě). Nejvýznamnější bit je připojen na jeden ze dvou výstupních vývodů v závislosti na volbě mezi 2drátovým nebo 3drátovým režimem. Průchozí latch je vložen mezi výstup sériového registru a výstupní vývod. Zajišťuje se tak synchronizace změny dat výstupu na opačnou hranu hodin než je ta, kterou se vzorkují vstupní data. Sériový vstup je vždy vzorkován na vývodu **DI** nezávisle na konfiguraci jednotky USI.

Do 4bitového čítače může být jak zapisováno, tak lze číst jeho obsah přes datovou sběrnici. Též může generovat přerušení při přetečení. Jak sériový registr tak i čítač jsou řízeny shodným hodinovým signálem. To dovoluje, aby čítač čítal počet bitů přijatých nebo vyslaných a generoval přerušení, když je přenos kompletní. Poznamenejme, že když je jako zdroj hodin vybrán vnější hodinový signál, čítá čítač obě hrany hodin a ne počet bitů. Hodiny mohou být vybrány ze 3 odlišných zdrojů:

• vývod **USCK**,

• přetečení čítače/časovače 0,

• programově řízené čítání.

Řídicí hodiny 2drátového přenosu mohou generovat přerušení, když je detekován START stav. Také může generovat čekací stavy prodlužující „log. 0“ na hodinovém vývodu po detekci START stavu až do přetečení čítače.

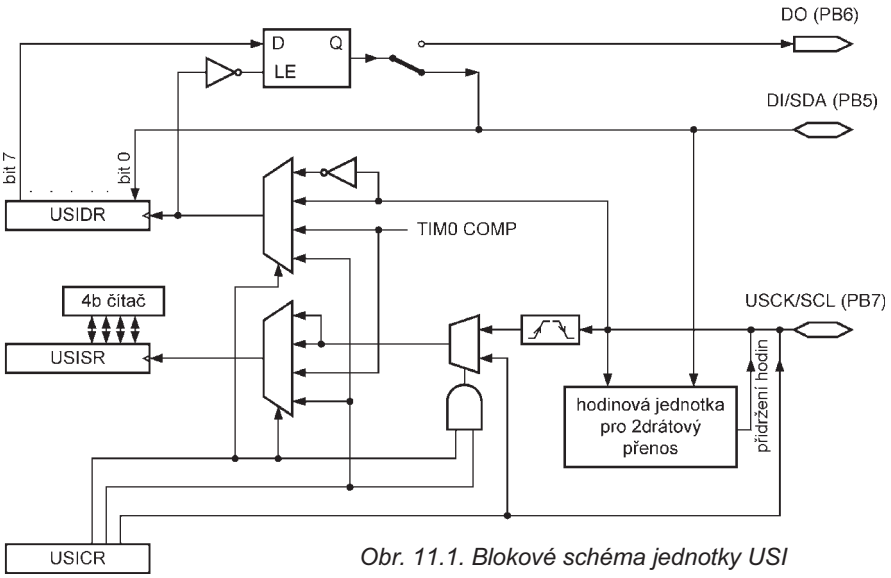
11.2. Funkční popis

Nyní budou vysvětleny jednotlivé režimy komunikace.

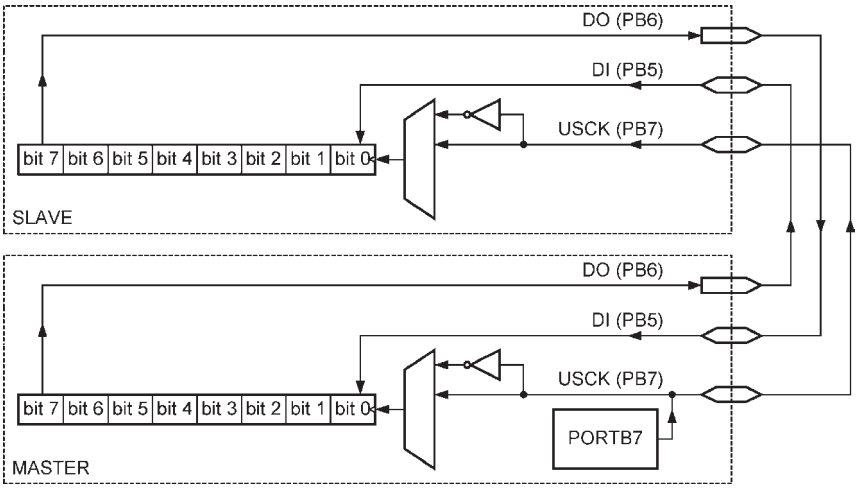
3drátový režim (SPI sběrnice)

3drátový režim jednotky USI odpovídá SPI režimu 0 nebo 1, nemá však funkcionalitu vývodu **SS** (Slave Select). Nicméně tento požadavek lze v případě nezbytnosti řešit programově. Jména vývodů používaná v tomto režimu jsou: **DI (vstupní data)**, **DO (výstupní data)**, **USCK (hodiny přenosu)**.

Pozn.: SPI (Serial Peripheral Interface) označuje 3drátovou sériovou



Obr. 11.1. Blokové schéma jednotky USI



Obr. 11.2. Propojení jednotek master a slave při 3drátové komunikaci

sběrnici vybavenou signály: hodiny, vstupní data a výstupní data. Jedná se o sběrnici, která má jeden nadřazený obvod (master), který generuje hodinový signál a dále libovolný počet podřízených obvodů (slave). Pro výběr podřízeného obvodu obvykle slouží vývod SS (v případě jednotky USI lze pro tento účel použít libovolný vývod mikrořadiče, který budeme ovládat přímo programově).

Na obr. 11.2 jsou znázorněny dvě jednotky pracující ve 3drátovém režimu (master a slave). Dva posuvné registry jsou vzájemně propojeny, takže po 8 hodinových impulsech **USCK** se data v obou registrech vzájemně vymění. Stejný hodinový signál také inkrementuje obsah 4bitového čítače jednotky USI. Takže příznak přetečení čítače (**USIOIF**) pak může být použit k určení, že byl přenos dokončen.

Hodiny jsou generovány jednotkou master buď programovou změnou vývodu **USCK** (přes registr **PORTD**) nebo zápisem 1 do bitu **USITC** z registru **USICR**.

Časování ve 3drátovém režimu je znázorněno na obr. 11.3. Zcela nahoře jsou uvedeny referenční hodiny **USCK**. V každém hodinovém cyklu je do datového registru jednotky USI (**USIDR**) vsunut jeden bit.

Časování **USCK** je uvedeno pro oba režimy vnějších hodin. V režimu **USICS0 = 0** je vstup **DI** vzorkován náběžnou hranou a výstup **DO** se mění (bit je vysunut ven) sestupnou hranou. V režimu **USICS0 = 1** se hrany používají opačně (vzorkování je sestupnou hranou a výstup náběžnou hranou). Tyto režimy odpovídají režimům SPI 0 a SPI 1.

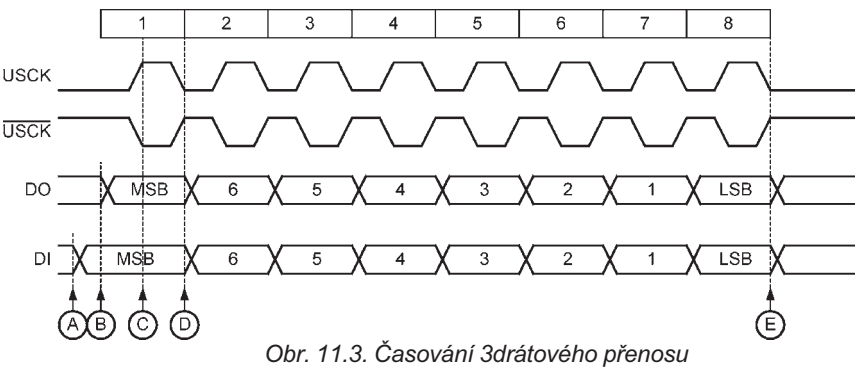
Podle obr. 11.3 probíhá přenos v těchto krocích:

1. Slave a master nastaví svůj datový výstup v závislosti na použitém protokolu, povolí svůj výstupní budič (značky **A** a **B**). Výstup je nastaven zápisem dat, která mají být přenesena do sériového datového registru. Povolení výstupu je provedeno nastavením odpovídajícího bitu registru **DDRB**. Poznamenejme, že body **A** a **B** nemají specifické pořadí, ale oba musí být umístěny minimálně do poloviny cyklu **USCK**, tedy před bodem **C** (zde se data vzorkují). 4bitový čítač je vynulován.

2. Master generuje hodinový puls dvojí programovou změnou stavu linky **USCK** (body **C** a **D**). Bitová hodnota datového vstupu **DI** slave a master je vzorkována v první hraně (**C**) a datový výstup je měněn opačnou hranou (**D**). 4bitový čítač bude čítat obě hrany.

3. Krok 2 se opakuje ještě 7× pro kompletní přenos bajtu.

4. Po 8 hodinových pulzech (resp. 16 hranách hodin) čítač přeteče a indikuje dokončení přenosu. Přenášené datové bajty musí být zpracovány před tím, než se začne nový přenos. Přerušení při přetečení probudí procesor, pokud se nacházel v režimu Idle. V závislosti na použitém protokolu může slave nyní uvést svůj výstup do stavu vysoké impedance.



Obr. 11.3. Časování 3drátového přenosu

Příklad SPI master operace

Následující kód demonstruje, jak použít jednotku USI jako SPI master:

```
SPITR:  OUT USIDR,R16      ;ulož data z R16
        LDI R16,(1<<USIOIF)
        OUT USISR,R16      ;vynuluj príznak USIOIF
        LDI R16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC)
SPITL:  OUT USICR,R16      ;odešli jeden bit
        SBIS USISR,USIOIF  ;test konce prenosu
        RJMP SPITL         ;prenos nedokoncen-
                           ;zpet do smycky
        IN R16,USIDR       ;ulož data do R16
        RET
```

Uvedený kód je optimalizován na co nejkratší délku. Předpokládá se, že vývody **DO** a **USCK** jsou nastaveny jako výstupy (přes registr **DDRB**). Před voláním dané rutiny musí být do registru **R16** vložena vysílaná hodnota. Po dokončení přenosu je nově přečtená hodnota k dispozici v registru **R16**.

V datasheetu mikrořadiče ATtiny2313 na str. 141 najdeme další příklad master operace (optimalizovaný na co nejvyšší rychlost).

Příklad SPI slave operace

Následující kód demonstruje, jak použít jednotku USI jako SPI slave:

```
INIT:   LDI R16,(1<<USIWM0)|(1<<USICS1)
                           ;konfigurace
        OUT USICR,R16
        ...
SSPIT:  OUT USIDR,R16      ;ulož data z R16
        LDI R16,(1<<USIOIF)
        OUT USISR,R16      ;nuluj príznak USIOIF
SSPIL:  SBIS USISR,USIOIF  ;test konce prenosu
        RJMP SSPIL
        IN R16,USIDR       ;ulož data do R16
        RET
```

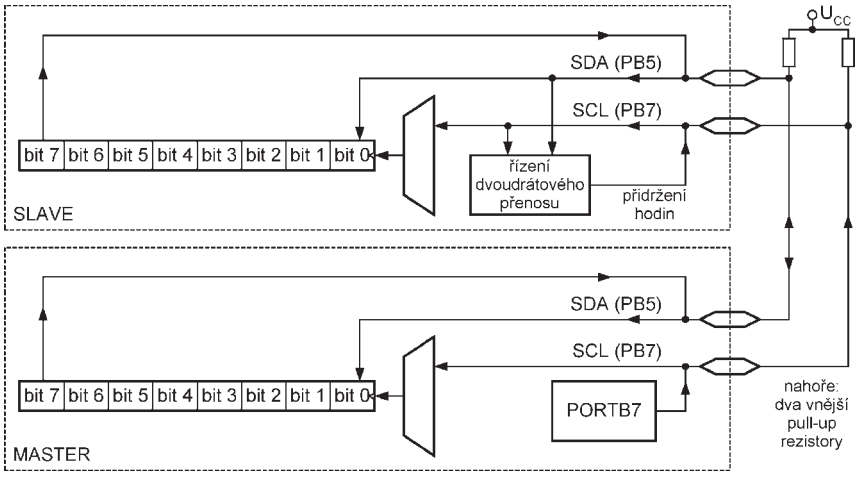
Uvedený kód je optimalizován na co nejkratší délku. Předpokládá se, že vývod **DO** je konfigurován jako výstup a vývod **USCK** jako vstup (přes registr **DDRB**). Před voláním rutiny **SSPIT** musí být do registru **R16** vložena vysílaná hodnota. Po dokončení přenosu je nově přečtená hodnota k dispozici v registru **R16**.

Rutina **INIT** slouží k inicializaci jednotky USI. Tyto instrukce nastaví 3drátový režim a pro vzorkování zvolí náběžnou hranu hodin.

2drátový režim (sběrnice I²C)

Ve 2drátovém režimu odpovídá jednotka USI rozhraní I²C. Nemá však zavedeno omezení rychlosti přeběhu výstupů a filtraci šumu na vstupech. Názvy vývodů jsou **SDA (datová linka)** a **SCL (hodiny)**.

Na obr. 11.4 je ukázáno, jak jednotka USI pracuje ve 2drátovém režimu. Hlavní odlišnost mezi operacemi master a slave je v použití jednotky řízení hodin. Generování hodin musí být implementováno programově, vyslání jednotlivých bitů se provede automaticky.



Obr. 11.4. Propojení jednotek master a slave při 2drátové komunikaci

Pro posouvání dat se používá sestupná hrana.

Slave může vložit čekací stavy na začátek nebo na konec přenosu tak, že prostě přidrží linku **SCL** v „log. 0“. To značí, že master musí testovat linku **SCL**, zda je aktuálně uvolněna, a teprve potom může vytvořit náběžnou hranu hodin.

Vzhledem k tomu, že hodiny také inkrementují čítač, lze přetečení čítače použít k indikaci dokončení přenosu. Hodiny jsou generovány masterem negací vývodu **SCL** pomocí registru **PORTB**.

Směr toku dat není dán fyzickou vrstvou, řízení toku dat je dáno použitým protokolem (závisí na konkrétních obvodech).

Připomeňme ve zkratce, jak vlastně funguje sběrnice I^2C (viz obr. 11.5 a lit. [3]):

- Přenos začíná tzv. START stavem. Sestupná hrana linky SDA při SCL = 1.
- Každé zařízení má svou jedinečnou 7bitovou adresu, která je doplněna bitem určujícím směr přenosu (čtení/zápis). Proto je možno připojovat více obvodů na stejné linky.
- Po přenosu každého bajtu musí podřízený obvod stáhnout linku SDA do „log. 0“ (potvrzení - ACK). Pokud podřízený obvod chce indikovat, že data nejsou určena pro něj, ponechá v místě tohoto 9. bitu linku SDA v „log. 1“ (NACK).
- Po naadresování následuje vysílání dat, v místě 9. bitu bude opět vložen potvrzovací příznak ACK.
- Přenos končí tzv. STOP stavem. Náběžnou hranou linky SDA při SCL = 1.

S ohledem na obr. 11.5 musí přenos proběhnout v těchto krocích:

1. START stav je generován masterem tak, že SDA přejde do log. 0 v okamžiku, když je SCL = 1 (**A**). SDA můžeme do „log. 0“ nastavit tak, že zapíšeme 0 do 7. bitu posuvného registru nebo že přímo vynuluje bit **PORTB5**. Poznamenejme, že registr DDRB musí mít nastaven bit **DDRB5**, aby byla povolena funkce výstupu. Slave obvod startuje detekční logiku a nastaví příznak **USISIF**. Tento příznak může generovat přerušení, pokud je to nezbytné.

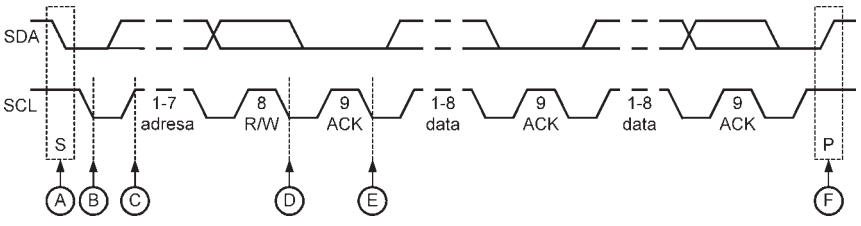
2. Detektor startu bude držet linku SCL v „log. 0“ po sestupné hraně způsobené masterem (**B**), takže obvod slave se probudí z režimu spánku nebo dokončí probíhající operaci před příjmem adresy. To je zajištěno vynulováním příznaku START stavu a vynulováním čítače.

3. Master nastaví první bit, který se má přenášet a uvolní linku SCL (**C**). Slave vzorkuje data a vsouvá je do sériového registru náběžnou hranou hodin SCL.

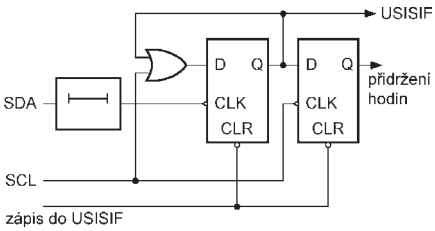
4. Po přenesení 8 bitů obsahuje slave obvod adresu a směr dat (čtení/zápis), čítač slave obvodu přeteče a SCL linka je držena v „log. 0“ (**D**). Pokud slave nebyl masterem neadresován, uvolní linku SCL a čeká na nový START stav.

5. Je-li slave naadresován, drží linku SDA v „log. 0“ v průběhu potvrzovacího cyklu před tím, než bude SCL držen v „log. 0“ (tedy registr čítače musí být nastaven na 14 před uvolněním SCL v bodě **D**). V závislosti na hodnotě bitu R/W master nebo slave povolí svůj výstup. Je-li tento bit nastaven je prováděna čtecí operace mastera (tedy slave řídí linku SDA). Slave může držet linku SCL po potvrzení (**E**).

6. Nyní může být ve stejném směru přeneseno několik bajtů, dokud nenastane



Obr. 11.5. Typické časování 2drátového přenosu



Obr. 11.6. Schéma zapojení detektoru stavu START

STOP stav daný masterem (**F**). Nebo může být vložen nový START stav.

Není-li slave schopen přijmout více dat, provádí NACK stav bajtu, který byl naposledy přijat. Když master provádí čtecí operaci, musí ji přerušit stažením linky SDA do log. 0 po odeslání posledního bajtu.

Detektor stavu START

Schéma zapojení detektoru stavu START je na obr. 11.6.

Linka SDA je zpožděna (v rozsahu 50 až 300 ns) pro zajištění platného vzorkování linky SCL. Detektor stavu START pracuje asynchronně a může procesor probudit z režimu spánku. Nicméně komunikační protokol může stanovit určité restrikce přesahu signálu linky SCL.

Alternativní použití jednotky USI

Pokud není jednotka USI používána pro sériovou komunikaci, může být (vzhledem ke své flexibilnosti) nastavena do alternativní funkce:

- **Asynchronní přenos dat s poloviční duplexem** - použitím posuvného registru ve 3drátovém režimu je možné implementovat více kompaktní a rychlou jednotku UART, která je řízená programově.
- **4bitový čítač** - 4bitový čítač může být použit jako standardní čítač s přerušením při přetečení. Poznamenejme, že když je čítač řízen vnějším signálem, zvyšují jeho obsah obě hrany hodin!
- **12bitový čítač/časovač** - kombinací 4bitového čítače a čítače/časovače 0 lze vytvořit 12bitový čítač.
- **Hranou spouštěné vnější přerušení** - po nastavení čítače na maximální hodnotu (1111) jej můžeme použít jako přídavný vstup vnějšího přerušení. Příznak přetečení a bit povolení přerušení jsou pak použity pro vnější přerušení. Tato schopnost se aktivuje bitem **USICS1**.
- **Softwarové přerušení** - přerušení při přetečení čítače může být použito jako programové přerušení spouštěné hodinovým vstupem.

11.3. Popis registrů jednotky USI

Nyní budou popsány jednotlivé registry, které slouží pro řízení jednotky USI.

*Obr. 11.7. Registr **USIDR***

Bit	7	6	5	4	3	2	1	0
	USIDR7	USIDR6	USIDR5	USIDR4	USIDR3	USIDR2	USIDR1	USIDR0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	?	?	?	?	?	?	?	?

*Obr. 11.8. Registr **USISR***

Bit	7	6	5	4	3	2	1	0
	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0
Čtení/zápis	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	0	0	0	0

*Obr. 11.9. Registr **USICR***

Bit	7	6	5	4	3	2	1	0
	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	W	W
Výchozí hodnota	0	0	0	0	0	0	0	0

USIDR - datový registr jednotky USI

Jednotka USI používá nebufferovaný sériový registr (obr. 11.7). Takže když přistupuje do registru **USIDR**, přistupuje vlastně přímo do sériového posuvného registru. Pokud se sériový hodinový impuls objeví ve stejném cyklu, kdy je zapisováno do registru, bude registr obsahovat zapisovanou hodnotu a nebude proveden žádný posuv. Posuv (doleva) je proveden v závislosti na nastavení bitů **USICS1**, **USICS0**. Posuv může být řízen vnějším hodinovým pulzem, přetečením čítače/časovače 0 nebo přímo programově použitím strobovacího bitu **USICLK**. Poznamenejme, že i když není vybrán žádný režim zápisu (**USIWM1,0** = 00), jak vnější vstupy (**DI/SDA**), tak vnější hodinový vstup (**USCK/SCL**), mohou být stále používány posuvným registrem.

Používané výstupní vývody (**DO/SDA** v závislosti na režimu zápisu) jsou připojeny přes výstupní latch (klopný obvod typu D řízený úrovní) na nejvýznamnější bit (bit 7) datového registru. V případě, že je zvolen vnější hodinový zdroj (**USICS1** = 1), je výstupní latch otevřen (tedy je průchozí) v průběhu první poloviny cyklu sériových hodin. V případě, že je vybrán vnitřní hodinový zdroj (**USICS1** = 0), bude výstup změněn okamžitě, když je zapsán nový nejvýznamnější bit (dokud je latch otevřen). Latch zajišťuje, že datový vstup je vzorkován opačnou hranou hodin než je ta, kterou se vzorkuje vstup. Poznamenejme, že odpovídající bit registru **DDRB** musí být nastaven proto, aby se povolil výstup sériového registru.

USISR - stavový registr jednotky USI

Stavový registr obsahuje příznaky přerušení, stavy linky a obsah čítače (obr. 11.8).

Význam jednotlivých bitů:

• **USISIF (příznak přerušení START stavu)** - je-li vybrán 2drátový režim, je příznak **USISIF** nastaven (1), když je detekován START stav. Při odstavení výstupu nebo ve 3drátovém režimu a při **USICLK** = 0 a **USICS1** = 1 nastaví tento příznak libovolná hrana vývodu SCL. Přerušení bude generováno, když je tento příznak nastaven v okamžiku nastavení bitů **USISIE** (z registru **USICR**) a **I** (z registru **SREG**). Tento příznak lze nulovat pouze zápisem 1 do **USISIF**. Vynulování pak uvolní detekci START stavu. Přerušení způsobené START stavem probouzí procesor ze všech režimů snížené spotřeby.

• **USIDIF (příznak přerušení při přetečení čítače)** - tento příznak je nastaven při přetečení 4bitového čítače (přechod 1111 na 0000). Přerušení bude generováno, pokud jsou nastaveny bity **USIOIE** (z registru **USICR**) a **I** (z registru **SREG**). Tento příznak lze vynulovat pouze zápisem 1 do **USIOIF**. Přerušení způsobené přetečením čítače probouzí procesor z režimu idle.

• **USIPF (příznak STOP stavu)** - ve 2drátovém režimu je příznak **USIPF** nastaven, když je detekován stav STOP. Tento příznak lze vynulovat pouze zápisem 1 do **USIPF**. Poznamenejme, že ten-to příznak neslouží pro spuštění přerušení, lze jej použít pouze pro řízení přenosu.

• **USIDC (příznak kolize datového výstupu)** - příznak **USIDC** se nastaví, pokud se bit 7 posuvného registru liší od aktuální hodnoty vývodu. Tento příznak je platný pouze ve 2drátovém režimu. Poznamenejme, že tento příznak neslouží pro spuštění přerušení, lze jej použít pouze pro řízení přenosu.

• **USICNT3 až USICNT0 (obsah čítače)** - tyto bity představují obsah 4bitového čítače. Obsah čítače lze číst/zapisovat přímo programově. Obsah čítače je inkrementován jedním z těchto zdrojů: detektorem hrany vnějších hodin, přetečením čítače/časovače 0, programově použitím bitů **USICLK** nebo **USITC**. Zdroj hodin závisí na nastavení bitů **USICS1** a **USICS0**. Při použití vnějších hodin lze hodiny generovat zápisem do strobovacího bitu **USITC**. Tato schopnost se povolí nastavením **USICLK** = 1 při **USICS1** = = 1. Poznamenejme, že i když není vybrán žádný režim (**USIWM1,0** = 00), může být vnější hodinový vstup (**USCK/SCL**) stále používán čítačem.

USICR - řídicí registr jednotky USI

Řídicí registr obsahuje bity pro povolení přerušení, volbu režimu, výběr hodin a volbu jejich hrany (obr. 11.9).

Význam jednotlivých bitů:

• **USISIE (povolení přerušení při stavu START)** - nastavení tohoto bitu

Tab. 11.1. Výběr režimu

USIWM1, USIWM0	Popis
00	Výstupy, podržení hodin a detekce startu jsou odstaveny. Vývody portu pracují normálně.
01	Zvolen 3drátový režim, který používá vývody **DO**, **DI** a **USCK**. • **Datový výstup (DO)** překrývá odpovídající bit registru **PORTB6**, nicméně odpovídající bit **DDRB6** stále řídí směr dat. Takže když je vývod portu konfigurován jako vstup, pull-up rezistor řízen bitem **PORTB6**. • **Datový vstup (DI)** a **sériové hodiny (USCK)** nemění normální funkci portu. Když jednotka pracuje jako master (vývod **PB7** je konfigurován jako výstup), jsou hodinové pulzy generovány programově negací registru **PORTB7**. Pro tento účel lze použít bit **USITC**.
10	Zvolen 2drátový režim, který používá vývody **SDA** a **SCL**. • **Sériová data (SDA)** a **sériové hodiny (SCL)** jsou obousměrné a používají budiče s otevřeným kolektorem. Výstupní budiče jsou povoleny nastavením odpovídajících bitů v registru **DDRB**. Když je povolen výstupní budič pro vývod SDA, bude výstupní budič uvádět linku SDA do log. 0 v případě, že je výstup posuvného registru nebo odpovídající bit **PORTB5** vynulován. Jinak je linka SDA uvolněna. Když je povolen výstupní budič pro linku SCL, bude linka uvedena do log. 0 v případě, že je vynulován odpovídající bit **PB7** nebo detektor START stavu. Jinak není linka SCL buzena. Linka SCL je držena v log. 0, když je detekován START stav a výstup povolen. Vynulováním bitu **USISIF** (příznak START stavu) je linka uvolněna. Pull-up rezistory na vývodech SDA a SCL jsou v tomto režimu odstaveny (musí být dodány z vnějšku).
11	Pracuje stejně jako režim 10. Rozdíl je v tom, že linka SCL je držena v úrovni „log. 0“ také tehdy, když přeteče 4bitový čítač až do vynulování příznaku **USIOIF**.

Tab. 11.2. Výběr hodinového zdroje

USICS1	USICS0	USICLK	Zdroj hodin pro posuv. registr	Zdroj hodin pro 4bitový čítač
0	0	0	žádné hodiny	žádné hodiny
0	0	1	programové strobování (**USICLK**)	programové strobování (**USICLK**)
0	1	X	přetečení č/č 0	přetečení č/č 0
1	0	0	vnější, vzestup. hrana	vnější, obě hrany
1	1	0	vnější, sestup. hrana	vnější, obě hrany
1	0	1	vnější, vzestup. hrana	programové strobování (**USICLK**)
1	1	1	vnější, sestup. hrana	programové strobování (**USICLK**)

povoluje přerušení při detekci stavu START.

• **USIOIE (povolení přerušení při přetečení čítače)** - nastavení tohoto bitu povoluje přerušení při přetečení 4bitového čítače.

• **USIWM1, USIWM0 (volba režimu)** - tyto bity volí 2drátový nebo 3drátový režim (viz tab. 11.1).

• **USICS1, USICS0 (výběr zdroje hodin)** - tyto bity volí hodinový zdroj (viz tab. 11.2) pro posuvný registr a 4bitový čítač. Výstupní latch zajišťuje, že při použití vnějšího hodinového zdroje (**USCK/SCI**) probíhá změna výstupu opačnou hranou než vzorkování vstupu (**DI/SDA**). Při volbě programového spouštění nebo použití čítače/časovače 0 je výstupní latch průchozí, a proto se výstup mění okamžitě. Pro **USICS1,0** = 00 je navoleno programové strobování, které probíhá zápisem 1 do bitu **USICLK** (jak pro posuvný registr, tak i pro čítač). Při volbě vnějšího zdroje hodin (**USICS1** = 1) slouží bit **USICLK** pro volbu strobováním z vnějšího zdroje nebo programové strobování bitem **USITC**.

• **USICLK (strobování hodin)** - zápis 1 do bitu **USICLK** způsobí posuv posuvného registr o jeden krok a současné zvýšení obsahu 4bitového čítače o 1 (pokud je zvoleno programové strobování, **USICS1,0** = 00). Výstup se pak mění okamžitě ve stejném hodinovém cyklu. Hodnota vsunutá do posuvného registru je vzorkována v předchozím instrukčním cyklu. Při volbě vnějšího hodinového zdroje (**USICS1** = 1) se funkce bitu **USICLK** mění, nastavení tohoto bitu aktivuje strobování bitem **USITC**.

• **USITC (negace hodinového vývodu)** - zápis 1 do bitu **USITC** způsobí negaci stavu linky **USCK/SCL** (z 1 na 0 nebo z 0 na 1). Pro povolení výstupu musí být nastaven bit **DDRB7**. Tato schopnost dovoluje snadné generování hodin implementované masterem. Při volbě vnějšího zdroje hodin (**USICS1** = 1) a nastavení **USICLK** = 1 způsobí zápis do **USITC** přímé strobování hodin pro 4bitový čítač. To dovoluje včasnou detekci dokončení přenosu, pokud zařízení pracuje jako master.

11.4. Přípravek ATUSIDSP - 3segmentový displej řízený 3drátovou sběrnicí

Kvůli vyzkoušení funkce 3drátového režimu poskytovaného jednotkou USI vznikl vcelku jednoduchý přípravek **ATUSIDSP**. Tento přípravek představuje displej se třemi segmenty. Schéma přípravku je na obr. 11.10.

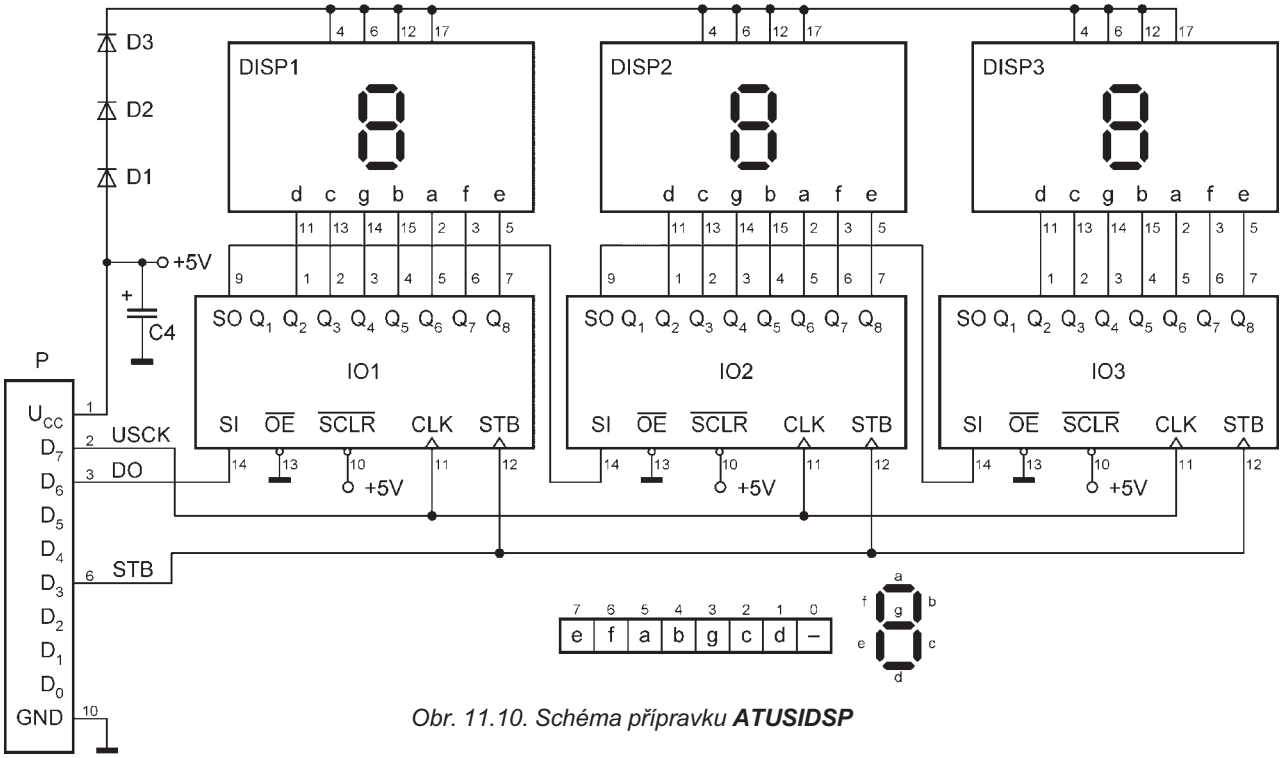
Při realizaci 3místného displeje ovládaného sériovou sběrnicí byly použity 3 obvody (IO1 až IO3) typu **74HCT595**.

Určitým problémem bylo omezit výstupní proud těchto obvodů tak, aby nebyly proudově nebo výkonově přetěžovány. Použití omezovacích rezistorů by bylo nevhodné (zvětšení rozměrů přípravku, složitější konstrukce). Nakonec byl výstupní proud omezen snížením anodového napětí displejů pomocí tří běžných diod, které vyhověly celkovým odebíraným proudem (1N4001 až 1N4007).

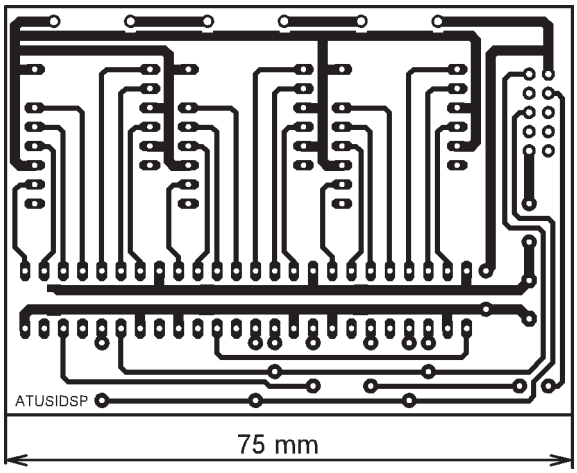
Vstup sériových dat (**SI**) prvního obvodu je připojen na výstup jednotky USI (**DO/PB6**), hodinový signál (**CLK**) všech obvodů je pak připojen na hodinový signál jednotky USI (**USCK**). Jednotlivé obvody tvoří kaskádně zapojený registr, takže výstup **SO** obvodu **IO1** je připojen na vstup **SI** obvodu **IO2**. Podobně výstup **SO** obvodu **IO2** je připojen na vstup **SI** obvodu **IO3**.

Přepis sériově přijatých dat z vnitřních sériových registrů do výstupního budiče je řízen linkou **PB3** (na straně obvodů IO1 až IO3 se jedná o signál **STB**). Přepis proběhne náběžnou hranou tohoto signálu.

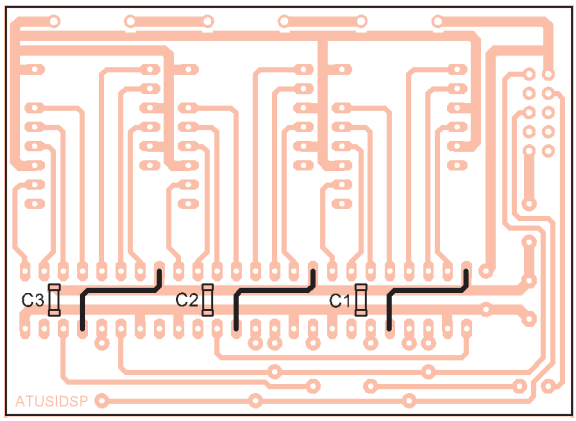
S ohledem na zjednodušení plošných spojů nebyly jednotlivé segmenty připojeny popořadě. Takže např. nejvyšší bit **Q8** odpovídá segmentu **e**. Desetinná tečka není ovladatelná (nelze



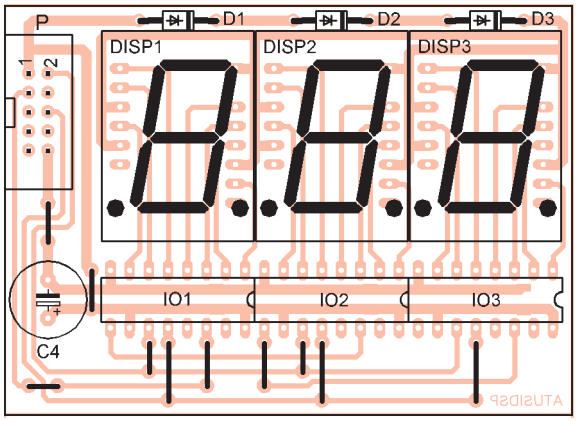
*Obr. 11.10. Schéma přípravku **ATUSIDSP***



Obr. 11.11. Obrazec plošných spojů přípravku ATUSIDSP (měř.: 1 : 1)



Obr. 11.12 (vpravo nahoře). Rozmístění součástek SMD na straně spojů na desce přípravku ATUSIDSP



Obr. 11.13 (vpravo dole). Rozmístění vývodových součástek na desce přípravku ATUSIDSP

ji rozsvítit). Situace je zřejmá z obr. 11.10.

Kondenzátor **C4** blokuje napájecí napětí. Kromě toho jsou na přípravku ještě tři blokovací kondenzátory (C1 až C3), které však nejsou pro jednoduchost zakresleny (každý z těchto kondenzátorů blokuje napájecí napětí jednoho posuvného registru). Rovněž není zakresleno připojení napájecího napětí k obvodů IO1 až IO3.

Všechny součástky přípravku jesou připájeny na desku s jednostrannými plošnými spoji.

Vzhledem ke komplikacím, které vznikly ze snah minimalizovat rozměry desky a současně se vyhnout použití oboustranné desky, bylo nutné použít několik triků.

Předně - deska se osazuje i ze strany spojů - viz obr. 11.12. Ze strany spojů je třeba zapájet tři blokovací kondenzátory C1 až C3 v provedení SMD. Dále jsou na straně spojů vedeny tři drátové propojky izolovanými dráty.

Ze strany součástek je třeba zapájet 10 drátových propojek.

Výkresy potřebné pro zhotovení desky s plošnými spoji jsou na obr. 11.11 až obr. 11.13.

Seznam součástek pro ATUSIDSP

(cena asi 200 Kč)

C1 až C3	100 nF/X7R, SMD 1206
C4	470 µF/16 V, radiální
D1 až D3	1N4007
IO1 až IO3	74HCT595 (74HC595, 74LS595)
DISP1 až DISP3	HDSP-8601 (společné anody)
P	MLW10G

deska s plošnými spoji ATUSIDSP

11.5. Příklady použití 3drátového režimu

Dále jsou uvedeny dva příklady použití 3drátového režimu, tedy vlastně přípravku **ATUSIDSP**.

První příklad uvádí základní rutiny pro ovládání displeje:

- První rutina (**SPITR**) slouží pro odeslání jednoho bajtu sběrnicí USI (což odpovídá jednomu segmentu displeje).
- Druhá rutina (**DISP1**) chápe obsah registru R16 jako desítkovou číslici. Pomocí tabulky (**TAB**) kombinací, které odpovídají segmentům jednotlivých číslic, převede číslici na odpovídají stav displeje. Tuto hodnotu pak vyvoláním rutiny **SPITR** odešle na displej.

Druhý příklad je vlastně rozšířením předchozího příkladu. Cyklicky zobrazuje číselné kombinace 000 až 999.

Rozdělení na dva příklady bylo provedeno pro zvýšení přehlednosti.

Zadání příkladu 21: Připojte k vývojovému kitu **SDK2313** na port B přípravek **ATUSIDSP**. Vytvořte tabulku pro zobrazení desítkových číslic (0 až 9) a rozsviťte na displeji nápis 123.

V programu je nejdříve definován symbol **STB** (označuje třetí bit portu, na který je připojen přípravek **ATUSIDSP**. Dále je nutno konfigurovat vývody **USCK**, **DO** a **STB** jako výstupní (vhodným nastavením registru **DDRB**). Linka **STB** bude zpočátku v „log. 0“.

Dále se konfiguruje jednotka USI. Po předchozí inicializaci musíme na displej poslat žádanou kombinaci 123 ovšem v pořadí 3, 2, 1. Opačné pořadí je nutné proto, že nejdříve odeslaný bajt se postupně dostane až na místo posledního segmentu. Pro odesílání slouží rutina **DISP1**.

Po odeslání všech tří číslic musí být vložen hodinový impuls na linku **STB** (**SBI PORTB,STB** a **CBI PORT,STB**) a potom je program zacyklen (instrukcí **RJMP PC**).

Rutina SPITR

Rutina **SPITR** bere obsah registru **R16** jako hodnotu, kterou chceme poslat sběrnicí USI. Nejdříve se vynuluje příznak **USIOIF**.

Odesílání je řízeno programově, takže konfigurační režim spolu s bitem **USITC** provádí negaci linky **USCK**, tedy generuje hodiny, a zároveň se testuje příznak **USIOIF** (indikuje dokončení přenosu).

Pokud je příznak **USIOIF** nastaven, proběhlo odeslání všech osmi bitů. Zároveň je v registru **USIDR** přijatý bajt. Ten je jako výsledek rutiny uložen do registru **R16**.

Rutinu **SPITR** lze používat i pro jiné obvody kompatibilní s 3drátovým režimem.

Rutina DISP1

Rutina **DISP1** bere obsah registru **R16** jako číslici, kterou chceme odeslat na displej **ATUSIDSP**. Vlastní odeslání je pochopitelně řešeno výše uvedenou rutinou **SPITR**. Úkolem této rutiny je převést číslici na odpovídající kombinaci segmentů.

Jednotlivé kombinace jsou definovány tabulkou **TAB**, která je pomocí direktivy **.DB** uložena do programové paměti (Flash). Pro přístup do tabulky je třeba definovat symbol **ADRTAB**, který představuje dvojnásobek hodnoty **TAB**. To proto, že programová paměť se standardně adresuje po slovech. Ovšem mi potřebujeme adresovat jednotlivé bajty.

Převod číslice na údaj odpovídající tabulce provedeme tak, že vypočítáme adresu položky přičtením hodnoty **ADRTAB** a žádané číslice. Adresa **ADRTAB** je nejdříve nahrána do ukaza-

*Tab 11.3. Program **PROG_21.ASM***

```
PROG_21.ASM:
        .NOLIST
        .INCLUDE "tn2313def.inc"
        .LIST
        .EQU STB=3          ;STB signál
        .CSEG
        LDI R16,RAMEND      ;nastav
        OUT SPL,R16         ;SPL
        LDI R16,0B11001000 ;povol USCK,
        OUT DDRB,R16        ;DO, STB
        CBI PORTB,STB       ;STB=0

        ;konfigurace USI:
        LDI R16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)
        OUT USICR,R16
        LDI R16,(1<<USIOIF) ;vynuluj
        OUT USISR,R16       ;príznak USIOIF

        ;pošli 3, 2, 1:
        LDI R16,3           ;pošli
        RCALL DISP1         ;S1=3
        LDI R16,2           ;pošli
        RCALL DISP1         ;S2=2
        LDI R16,1           ;pošli
        RCALL DISP1         ;S3=1
        SBI PORTB,STB       ;STB=1
        CBI PORTB,STB       ;STB=0
        RJMP PC             ;zacyklení

        ;pošle R16 prekódovaný na segmenty:
DISP1:  LDI ZL,LOW(ADRTAB) ;ZL ukazuje
        LDI ZH,HIGH(ADRTAB) ;na TAB
        ADD ZL,R16          ;prictení
        LDI R16,0           ;obsahu
        ADC ZH,R16          ;reg. R16
        LPM R16,Z           ;ctení z TAB+R16
        RCALL SPITR         ;pošli
        RET

        ;pošle R16 jednotkou USI:
SPITR:  OUT USIDR,R16       ;ulož data z R16
        LDI R16,(1<<USIOIF) ;vynuluj
        OUT USISR,R16       ;príznak USIOIF
        LDI R16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)|(1<<USITC)
SPITL:  OUT USICR,R16       ;odešli jeden bit
        SBIS USISR,USIOIF   ;konec prenosu?
        RJMP SPITL          ;prenos nedokoncen
        IN R16,USIDR        ;ulož data do R16
        RET

TAB:    .DB 0b00001001,0b11101011   ;0,1
        .DB 0b01000101,0b11000001   ;2,3
        .DB 0b10100011,0b10010001   ;4,5
        .DB 0b00010001,0b11001011   ;6,7
        .DB 0b00000001,0b10000001   ;8,9
        .EQU ADRTAB=2*TAB           ;adresa TAB
```

*Tab 11.4. Program **PROG_22.ASM***

```
PROG_22.ASM:
        .NOLIST
        .INCLUDE "tn2313def.inc"
        .LIST
        .EQU STB=3
        .DEF CITAC1=R18     ;registry
        .DEF CITAC2=R19     ;pro casovací
        .DEF CITAC3=R20     ;rutinu CEKEJ

        .CSEG
        LDI R16,RAMEND      ;nastav
        OUT SPL,R16         ;SPL
        LDI R16,0B11001000 ;povol USCK
        OUT DDRB,R16        ;DO, STB
        CBI PORTB,STB       ;STB=0

        ;konfigurace:
        LDI R16,(1<<USIWM0)|(1<<USICS1)|(1<<USICLK)
        OUT USICR,R16       ;USI
        LDI R16,(1<<USIOIF) ;vynuluj
        OUT USISR,R16       ;príznak USIOIF
        LDI R16,0           ;nuluj
        STS S1,R16          ;S1
        LDI R16,0           ;nuluj
        STS S2,R16          ;S2
        LDI R16,0           ;nuluj
        STS S3,R16          ;S3
SMYCKA: RCALL DISPC         ;zobraz stav
        LDS R16,S3          ;cti S3
        INC R16             ;zvyš o 1
        MOV R0,R16          ;kopie do R0
        CPI R16,10          ;test pretecení
        LDI R17,0           ;R17=0
        BRNE SEG2           ;nepreteklo->na SEG2
        CLR R0              ;preteklo->R0=0
        LDI R17,1           ;R17=1
SEG2:   STS S3,R0           ;ulož S3
        LDS R16,S2          ;cti S2
        ADD R16,R17         ;prenos z S3
        MOV R0,R16          ;kopie do R0
        CPI R16,10          ;test pretecení
        LDI R17,0           ;R17=0
        BRNE SEG1           ;nepreteklo->na SEG1
        CLR R0              ;preteklo->R0=0
        LDI R17,1           ;R17=1
SEG1:   STS S2,R0           ;ulož S2
        LDS R16,S1          ;cti S1
        ADD R16,R17         ;prenos z S2
        MOV R0,R16          ;kopie do R0
        CPI R16,10          ;test pretecení
        BRNE SEGV           ;nepreteklo->na SEGV
        CLR R0              ;preteklo->R0=0
SEGV:   STS S1,R0           ;ulož S1
        RCALL CEKEJ         ;zpoždení
        RJMP SMYCKA         ;aktualizace

        ;pošle S3, S2, S1 na displej:
DISPC:  LDS R16,S3
        RCALL DISP1         ;pošli S3
        LDS R16,S2
        RCALL DISP1         ;pošli S2
        LDS R16,S1
        RCALL DISP1         ;pošli S1
        SBI PORTB,STB       ;STB=1
        CBI PORTB,STB       ;STB=0
        RET

        ...
        ;rutiny DISP1 a SPITR
        ;viz PROG_21
        ...

        ;cekací rutina
CEKEJ:  LDI CITAC1,5        ;CITAC1=5
        CLR CITAC2          ;CITAC2=0
        CLR CITAC3          ;CITAC3=0
CEKEJA: DEC CITAC3
        BRNE CEKEJA         ;smycka 1
        DEC CITAC2
        BRNE CEKEJA         ;smycka 2
        DEC CITAC1
        BRNE CEKEJA         ;smycka 3
        RET                 ;návrat

TAB:    .DB 0b00001001,0b11101011   ;0,1
        .DB 0b01000101,0b11000001   ;2,3
        .DB 0b10100011,0b10010001   ;4,5
        .DB 0b00010001,0b11001011   ;6,7
        .DB 0b00000001,0b10000001   ;8,9
        .EQU ADRTAB=2*TAB

        .DSEG
S1:     .BYTE 1
S2:     .BYTE 1
S3:     .BYTE 1
```

tele **Z**. Potom je instrukcí **ADD ZL,R16** přičtena ke spodní části původní adresy požadovaná číslice. Následně je instrukcí **ADC ZH,R16** (mezitím byl registr **R16** vynulován) zohledněn případný přenos do vyšší poloviny adresy.

Instrukce **LPM R16,Z** již ze správně vypočítané adresy „vytáhne" odpovídající kombinaci pro segmentovku a uloží ji do registru **R16**. Následně vyvolaná rutina **SPITR** hodnotu odešle na displej.

Program k příkladu 21 je nazván **PROG_21.ASM** a je vypsán v tab. 11.3. Příklad najdete v [14] v adresáři **PROGRAMY\PROG_21**.

Zadání příkladu 22: Připojte k vývojovému kitu **SDK2313** na port B přípravek **ATUSIDSP**. Zobrazujte na displeji postupně čísla: 000, 001, 002 až 999.

Program vychází z příkladu **PROG_21**. Inicializace je téměř stejná jako v předchozím příkladu. Navíc je zařazeno vynulování proměnných, které jsou označeny jako **S1**, **S2**, **S3**. Tyto proměnné udržují stav jednotlivých segmentovek.

Zobrazení aktuálního stavu probíhá pomocí rutiny **DISPC**, která vlastně jen postupně čte hodnoty uložené v **S3 až S1** a odesílá je na přípravek **ATUSIDSP**. Po odeslání všech tří číslic vytvoříme na vývodu **STB** hodinový impuls.

Po zobrazení aktuálního stavu je nutné přestavit obsah proměnných **S3 až S1** tak, aby odpovídal následujícímu taktu. Tříciferné desítkové číslo se musí zvýšit o 1.

Nejdříve je tedy nahrán obsah proměnné **S3** (řád jednotek) do registru **R16** a následně zvýšen. Takto zvýšená hodnota je uložena do registru **R0**. Následně se testuje přetečení (tedy stav, kdy je R16 = 10). Pokud k přetečení nedošlo, je registr **R17** vynulován a zvýšená hodnota obsažená v registru **R0** se uloží zpět do **S3**. V opačném případě obsahuje registr **R17** hodnotu **1** (použije se jako přenos do vyššího řádu). Takže při přičítání 1 do dalšího řádu se navíc provede instrukce **ADD R16,R17**, která případný přenos uváží.

Přestavení proměnných **S2** a **S1** probíhá velmi podobně.

Po zobrazení každého čísla je vložena krátká čekací rutina.

Program k příkladu 22 je nazván **PROG_22.ASM** a je vypsán v tab. 11.4. Příklad najdete v [14] v adresáři **PROGRAMY\PROG_22**.

12. Mapa paměti a popis registrů

V této kapitole je popsáno rozdělení paměťového prostoru mikrořadiče **ATtiny2313** a dále je zde stručně připomenuta funkce jeho jednotlivých registrů.

12.1. Mapa paměti

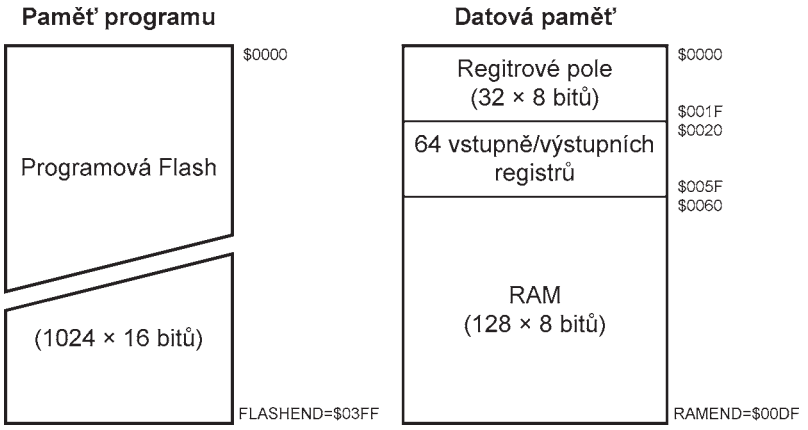
Procesory řady AVR používají **harvardskou architekturu**, která se vyznačuje odděleným paměťovým prostorem pro data a program.

Vnitřní programová paměť (Flash)

Do této paměti se ukládá program, který má být mikrořadičem vykonáván.

Vnitřní programová paměť je představována **programovou Flash**. Tato paměť je z hlediska probíhajícího programu adresována po slovech (dvojicích bajtů) - viz obr. 12.1.

Kapacita vnitřní paměti je 1024 slov (tedy 2 KB). Většina instrukcí má délku



Obr. 12.1. Mapa vnitřní paměti

jednoho slova, takže do procesoru lze vložit program obsahující až 1024 instrukcí.

Vnitřní datová paměť

Vnitřní datová paměť slouží pro uložení dat používaných programem. Jsou zde také realizovány registry procesoru.

Z obr. 12.1 je zřejmé, že datová paměť začíná blokem 32 bajtů, které odpovídají registrům registrové pole (R0 až R31). Následuje 64 vstupně/výstupních registrů (řídí periferie), které u mikrořadiče **ATtiny2313** nejsou všechny implementovány (viz kapitolu 12.2). Posledním blokem je libovolně použitelná paměť. Její kapacita je 128 bajtů.

Z obr. 12.1 je zřejmé, že každý registr registrového pole má svoji adresu v rozsahu 0 až 31, tedy se chová jako paměťová buňka (přístup přes instrukce pro práci s pamětí je však pomalejší). Nebudeme-li tedy používat všechny registry registrového pole, můžeme zbytek chápat jako paměťové buňky.

Vnitřní E^{2}PROM

Kromě datové a programové paměti disponují všechny mikrořadiče AVR vestavěnou datovou E^{2}PROM (E^{2}PROM = = EEPROM = Electrically Erasable PROM = elektricky vymazatelná PROM). Jedná se o paměť, která je nonvolativní, takže udržuje svůj obsah i při odpojení napájení (to je podstatný rozdíl oproti RAM). E^{2}PROM se maže elektricky přímo přístupem do určitých registrů AVR, takže není potřebný speciální programátor. Zaručený počet mazacích cyklů 100 000. Zápis do této paměti je časově náročnější, než do RAM (aby byl zápis úspěšný, musí se dodržet jistá sekvence instrukcí).

E^{2}PROM se nejčastěji používá k uložení konfiguračních dat, která zadá uživatel. Tato data se „neztratí" po vypnutí napájení. U mikrořadiče **ATtiny2313** je kapacita této paměti 128 bajtů.

Více o práci s E^{2}PROM je uvedeno v lit. [3].

12.2. Registry

Registry se rozdělují do dvou skupin. První je tzv. **registrové pole**, jedná se o 32 volně použitelných registrů označených jako **R0 až R31**. Druhou skupinou jsou **vstupně/výstupní registry**, které slouží k ovládání vestavěných periferií nebo samotného procesoru.

Registrové pole (Register File)

Registrové pole je tvořeno 32 všeobecně použitelnými registry délky 8 bitů. Všechny instrukce operující nad těmito registry mají přímý přístup ke všem aritmeticko-logickým instrukcím, který proběhne v rámci jediného hodinového cyklu. Výjimkou je 6 instrukcí, které pracují s konstantami: **SBCI**, **SUBI**, **CPI**, **ANDI**, **ORI**, **LDI**. Tyto instrukce jsou totiž použitelné jen pro horní polovinu registrového pole (R16 až R31).

Registrové pole obsazuje prvních 32 bajtů datové paměti. Protože se registrové pole nachází v datové paměti, lze ke každému z registrů přistupovat buď jeho jménem (R0 až R31) nebo jeho adresou ($0000 až $001F). Použití jména registru zajišťuje rychlý přístup, použití adresy dává možnost chápat registr jako paměťovou buňku.

Ukazatele (pointery)

Poslední registry registrového pole mohou být použity jako **16bitové ukazatele** při nepřímém adresování. Pomocí nepřímého adresování lze obsáhnout adresní prostor až 64 KB (2^{16} kombinací adresuje 65536 buněk paměti).

Poslední z těchto adresových ukazatelů lze rovněž použít jako adresový ukazatel při **tabulkové transformaci** (jedná se o nepřímé adresování programové paměti). Jména těchto registrů jsou **X**, **Y** a **Z** (registrové páry R26, R27; R28, R29 a R30, R31) - viz obr. 12.2.

Ukazatele slouží k nepřímému adresování paměti. Adresu buňky, se kterou chceme pracovat, uložíme do registru. Při volání instrukce přesunu dat se uvažuje jako zdroj nebo cíl buňka, jejíž adresa je uložena v ukazateli (ne tedy ukazatel samotný). Instrukční soubor disponuje možností pre-dekrementu a post-inkrementu, který je výborný při blokovému zpracování dat.

Při **blokovém zpracování dat** (např. při přesun bloku dat, hledání bajtu s určitou hodnotou apod.) se rovněž používají ukazatele. Do ukazatele uložíme adresu první buňky, se kterou chceme pracovat. Po každém dílčím kroku je třeba ukazatel posunout na další buňku.

Buď chceme pokračovat buňkou, která je na další adrese. Potom je výhodné využít **post-inkrement**. Jedná se o speciální variantu instrukce pracující s ukazatelem. Ta zajistí automatické zvýšení ukazatele po provedení instrukce (**post** značí po, **inkrement** znamená zvýšení o 1).

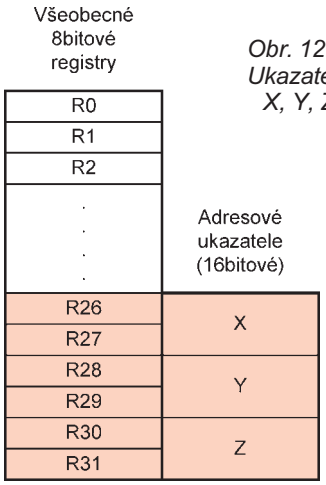
Jinou možností je, když chceme pokračovat buňkou na předchozí adrese. V tom případě je možno využít **pre-dekrement**. Opět je to speciální varianta instrukce, která zajistí snížení ukazatele před přístupem do paměti (**pre** značí před, **dekrement** znamená snížení o 1). Tato varianta je poněkud odlišná od post-inkrementu, obsah ukazatele se totiž změní ještě před přesunem (v některých případech tedy do ukazatele vložíme adresu o jednu vyšší).

Obrovskou výhodou post-inkrementu a pre-dekrementu je skutečnost, že modifikace ukazatele proběhne v rámci instrukce nepřímého adresování. Není tedy potřebný další hodinový impuls na vykonání instrukce, která by obsah ukazatele měnila „ručně".

Vstupně/výstupní registry (I/O Memory)

Vstupně/výstupní registry jsou patrně nejdůležitější částí mikrořadiče. Slouží totiž k řízení vestavěných periferií i samotného jádra mikrořadiče. Vstupně/výstupní registry jsou implementovány jako buňky datové paměti a jsou umístěny v prostoru $0020 až $005F. Všechny tyto registry jsou dostupné instrukcemi **IN** nebo **OUT** nebo jako paměťové buňky. Registry na adresách $20 až $3F lze ovládat/testovat i po bitech (pomocí instrukcí: **SBI, CBI, SBIS, SBIC**). Registry na adresách $40 až $5F takto ovládat/testovat nelze!

Umístění vstupně/výstupních registrů v datové paměti je přehledně uvedeno v tab. 12.1. Pro všechny registry lze používat instrukce **IN** nebo **OUT**. Pro šedě podbarvené registry lze navíc používat i instrukce **SBI, CBI, SBIS, SBIC**.



Obr. 12.2. Ukazatele X, Y, Z

Tab. 12.1. Vstupně/výstupní registry

Název	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SREG	I	T	H	S	V	N	Z	C
SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
OCR0B	čítač/časovač 0 – komparační registr B							
GIMSK	INT1	INT0	PCIE	–	–	–	–	–
EIFR	INTF1	INTF0	PCIF	–	–	–	–	–
TIMSK	TOIE1	OCIE1A	OCIE1B	–	ICIE1	OCIE0B	TOIE0	OCIE0A
TIFR	TOV1	OCF1A	OCF1B	–	ICF1	OCF0B	TOV0	OCF0A
SPMCSR	–	–	–	CTPB	RFLB	PGWRT	PGERS	SELFPRGEN
OCR0A	čítač/časovač 0 – komparační registr A							
MCUCR	PUD	SM1	SE	SM0	ISC11	ISC10	ISC01	ISC00
MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF
TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
TCNT0	čítač/časovač 0 – aktuální obsah							
OSCCAL	–	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0
TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10
TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
TCNT1H	čítač/časovač 1 – aktuální obsah (horní část)							
TCNT1L	čítač/časovač 1 – aktuální obsah (dolní část)							
OCR1AH	čítač/časovač 1 – komparační registr A (horní část)							
OCR1AL	čítač/časovač 1 – komparační registr A (dolní část)							
OCR1BH	čítač/časovač 1 – komparační registr B (horní část)							
OCR1BL	čítač/časovač 1 – komparační registr B (dolní část)							
CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
ICR1H	čítač/časovač 1 – záchytný registr (horní část)							
ICR1L	čítač/časovač 1 – záchytný registr (dolní část)							
GTCCR	–	–	–	–	–	–	–	PSR10
TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–
WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
PCMSK	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
EEAR	–	adresový registr E^{2}PROM						
EEDR	datový registr E^{2}PROM							
EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
PORTA	–	–	–	–	–	PORTA2	PORTA1	PORTA0
DDRA	–	–	–	–	–	DDA2	DDA1	DDA0
PINA	–	–	–	–	–	PINA2	PINA1	PINA0
PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
GPIOR2	obecně použitelný v/v registr 2							
GPIOR1	obecně použitelný v/v registr 1							
GPIOR0	obecně použitelný v/v registr 0							
PORTD	–	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
DDRD	–	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
PIND	–	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
USIDR	datový registr jednotky USI							
USISR	USISIF	USIOIF	USIPF	USIDC	USICNT3	USICNT2	USICNT1	USICNT0
USICR	USISIE	USIOIE	USIWM1	USIWM0	USICS1	USICS0	USICLK	USITC
UDR	datový registr jednotky USART							
UCSRA	RXC	TXC	UDRE	FE	DOR	UPE	U2X	MPCM
UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
UBRRL	registr přenosové rychlosti USART (dolní část)							
ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
UCSRC	–	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
UBRRH	–	–	–	–	registr přenosové rychlosti USART (horní část)			
DIDR	–	–	–	–	–	–	AIN1D	AIN0D

13. Instrukční soubor a assembler

Instrukční soubor je sada všech instrukcí, které procesor zná (umí je vykonat). Instrukce se obvykle rozdělují podle druhu operace, kterou vykonávají, takové rozdělení si provedeme i my.

Nejdříve se však zmíníme o typech operandů instrukcí. Tuto partii je třeba pochopit nejdříve, protože zde zavedené symboly budeme používat při výčtu možných forem každé instrukce.

13.1. Operandy instrukcí

Instrukční soubor mikrořadičů AVR rozlišuje několik druhů operandů.

Než se seznámíme s typy operandů, bude vhodné vysvětlit nebo připomenout některé základní pojmy.

Přímé adresování - tím rozumíme situaci, kdy adresa je v instrukci přímo uvedena. U procesorů AVR lze adresovat registr registrového pole, vstupně/výstupní registr, datovou a programovou paměť. Ve všech těchto případech uvedeme adresu při zápisu instrukce.

Nepřímé adresování - tím rozumíme situaci, kdy adresa není součástí instrukce. Tato adresa vstupuje do instrukce z některého ukazatelového registru (X, Y, Z). Tato možnost usnadňuje zápis sekvencí, které opakovaně přistupují k jedné buňce paměti nebo k bloku za sebou uložených buněk.

RAMEND - označuje poslední platnou adresu datové paměti. Pro mikrořadič ATtiny2313 je RAMEND = $00DF - viz kapitolu 12.

FLASHEND - označuje poslední platnou adresu programové paměti. Pro mikrořadič ATtiny2313 je FLASHEND = = $03FF - viz kapitolu 12.

E2END - označuje poslední platnou adresu E^{2}PROM. Pro mikrořadič ATtiny2313 je E2END = $7F - viz kapitolu 12.

13.2. Typy skoků

Instrukce skoků umožňují změnit řízení programu přechodem na novou adresu. U mikrořadičů řady AVR mohou být používány různě „dlouhé" adresy.

Podmíněné skoky (relativní vzdálenost -64 až +63)

Podmíněné skoky (**branchs**) pracují se 7bitovou relativní adresou. Podmíněné skoky se provedou pouze v případě, že je splněna určitá podmínka - např. nastavený příznak C. Adresa je relativní vůči obsahu programového čítače (PC). Skutečná adresa se totiž získá součtem obsahu PC a této hodnoty.

Relativní 7bitová adresa dostupná pro podmíněné skoky umožňuje provádět skoky pouze v rozsahu –64 až +63. Tedy max. o 64 zpět nebo o 63 dopředu.

Přeskoky (skips)

Jedním z prvků procesorů RISC jsou tzv. přeskoky. Jedná se o zvláštní typ podmíněného skoku. Je-li podmínka splněna, přeskočí se následující instrukce. Pokud podmínka není splněna, následující instrukce se vykoná. Skok se realizuje přičtením hodnoty 2 nebo 3 k obsahu PC (přeskok pro případ následující instrukce dlouhé 1 nebo 2 slova) případně zvýšením PC o 1 (přeskok se neprovede).

Relativní skoky (relativní vzdálenost -2048 až +2047)

Instrukce **RJMP** a **RCALL** pracují s 12bitovou relativní adresou. Adresa je relativní vůči obsahu programového čítače (PC). Skutečná adresa se totiž získá součtem obsahu PC a této hodnoty.

Relativní 12bitová adresa umožňuje provádět skoky pouze v rozsahu -2048 až +2047. Tedy maximálně o 2048 zpět nebo o 2047 dopředu.

Nepřímý skok (rozsah adres 0 až 65535)

Instrukční soubor mikrořadičů AVR disponuje dvěma instrukcemi pro nepřímý skok (**IJMP** a **ICALL**). V tomto případě je brán obsah registru Z jako 16bitová adresa skoku. Jedná se tedy o nepřímý skok (adresa není obsažena v instrukci, ale v registru Z).

13.3. Zavedené symboly

Než přistoupíme k popisu instrukcí, je třeba uvést symboly, které budeme

Tab 13.1. Zavedené symboly pro zápis operandů instrukcí

Symbol	Výraz
data6	6bitová data (0 až 63)
data8	8bitová data (0 až 255)
off6	6bitový ofset (0 až 63)
s	bit registru SREG (0 až 7)
b	bit registru registrového pole nebo vstupu/výstupu (0 až 7)
Rr	zdrojový registr registrového pole (není-li uvedeno jinak, je **r** v rozsahu 0 až 31)
Rd	cílový registr registrového pole (není-li uvedeno jinak, je **d** v rozsahu 0 až 31)
P	vstupně/výstupní registr (není-li uvedeno jinak, je **P** v rozsahu 0 až 63)
addr16	16bitová adresa buňky v datové paměti
rel7	relativní 7bitová adresa (–64 až +63)
rel12	relativní 7bitová adresa (–2048 až +2047)

Tab 13.2. Symboly zavedené pro výklad instrukcí

Symbol	Výraz
←	přiřazení
↔	výměna
[]	obsah paměťové buňky (adresa je uvedena v závorce)
.	bit registru (číslo bitu je uvedeno za tečkou)
low	nižší polovina
high	vyšší polovina
and	logický součin
or	logický součet
xor	výlučný logický součet

používat. V tab 13.1 jsou shrnuty zavedené symboly pro zápis instrukcí. V tab. 13.2 jsou zase symboly pro vysvětlení funkce instrukcí.

Instrukce jsou uvedeny v dalších tabulkách podle jejich zaměření.

13.4. Přesuny dat

Výklad instrukcí zahájíme nejobjemnější skupinou instrukcí přesunů dat (tab. 13.3), protože tyto instrukce jsou nejjednodušší na pochopení.

Tyto instrukce neovlivňují žádný z příznaků registru **SREG** (vyjma případu zápisu do registru **SREG** nebo práce s bity tohoto registru).

• Nejjednodušší formou je instrukce **MOV Rd,Rr**, která přesouvá data mezi dvěma registry registrového pole. Hodnota uložená v registru **Rr** (**r** jako sou**r**ce) se překopíruje do registru **Rd** (**d** jako **d**estination). Obsah registru **Rr** se pochopitelně nezmění.

Jako příklad můžeme uvést instrukci **MOV R10,R20**. Tato instrukce tedy obsah registru **R20** překopíruje do registru **R10**. Výsledkem je, že oba registry mají stejné obsahy (původní obsah registru **R20** se instrukcí nezměnil).

• Velmi často je používána instrukce **LDI Rd,data8**. Slouží k nastavení registru **Rd** přímou hodnotou (konstantou) **data8**. Do strojového kódu instrukce musí kromě operačního kódu „vejít" i číslo registru a 8 bitů představujících konstantu. Proto nelze adresovat celé registrové pole, ale pouze jeho horní část (**R16 až R31**). Pro **d** tedy platí, že je v rozsahu 16 až 31.

Jako příklad můžeme uvést instrukci **LDI R16,5**. Tato instrukce nastaví obsah registru **R16** na hodnotu **5**. Registr **R16** je prvním registrem, pro který můžeme instrukci **LDI** použít!

• Větší skupinu tvoří různé formy instrukce **LD**. Používají se pro nepřímé adresování datové paměti. Adresa je uložena v jednom z ukazatelů **X**, **Y**, **Z**. Ukazatel se může **po** resp. **před** přesunem posunout na následující resp. předchozí adresu. Hodnota přečtená z paměti se uloží do registru **Rd**.

Jako první příklad můžeme uvést instrukci **LD R0,X**. Takto vybereme z datové paměti z adresy odpovídající **X** (z místa kam ukazuje **X**) údaj a nahrajeme jej do registru **R0**.

Jako druhý příklad můžeme uvést instrukci **LD R0,X+**. Tato instrukce pracuje velmi podobně jako **LD R0,X**. Po provedení instrukce se obsah registru **X** zvýší o 1 (přesune se na následující adresu; ukazuje na následující adresu).

Jako třetí příklad můžeme uvést instrukci **LD R0,-X**. Tato instrukce nejdříve sníží obsah registru **X** o 1 (přesune se na předchozí adresu; ukazuje na předchozí adresu). Dále pak pracuje velmi podobně jako **LD R0,X**.

• Podobná je i instrukce **LDD**. Opět se jedná o nepřímé adresování pomocí ukazatelů **Y**, **Z**. Tato nepřímá adresa je doplněna přímým 6bitovým posunutím **off6** (tedy v rozsahu 0 až 63). Tímto způsobem lze adresovat data z tabulky, která v paměti začíná na adrese uložené v ukazateli. Posunutí pak určuje položku tabulky, kterou chceme nahrát. Hodnota přečtená z paměti se uloží do registru **Rd**.

Jako příklad můžeme uvést instrukci **LDS R5,Z+10**. Tato instrukce se obrátí do datové paměti na adresu danou součtem **Z+10**. Z této adresy překopíruje údaj do registru **R5**. Obsah registru **Z** se po vykonání instrukce **LDS** nemění.

• Instrukce **LDS Rd,addr16** se používá pro přímé adresování celého adresního prostoru datové paměti. Operand **addr16** uvádí plnou adresu datové buňky, jejíž hodnota se načte do registru **Rd**. Tato instrukce má délku dvou slov (32 bitová instrukce).

Jako příklad můžeme uvést instrukci **LDS R16,123**. Tato instrukce se obrátí do datové paměti na adresu **123**. Z této adresy překopíruje údaj do registru **R16**.

• Instrukce **ST** a **STD** jsou opakem instrukcí **LD** a **LDD**. Hodnota uložená v registru **Rr** se uloží do datové paměti na adresu, která je obsažena v ukazatelích **X**, **Y**, **Z**. Opět lze používat post-inkrement, pre-dekrement nebo přímé posunutí (to pouze u **Y** a **Z**).

• Instrukce **LPM** nahrává údaj z programové paměti (Flash). Adresa je opět určena nepřímo, obsahem registru **Z**. Údaj se nahraje do registru.

Jako první příklad můžeme uvést instrukci **LPM**. U této varianty je vše určeno „natvrdo", proto tato instrukce nemá operandy. Údaj z adresy v programové paměti, která je určena obsahem registru **Z**, se nahraje do registru **R0**.

Jako druhý příklad můžeme uvést instrukci **LPM R2,Z**. Tato instrukce pracuje velmi podobně jako **LPM**. Můžeme však volit cílový registr. Nyní je tedy údaj z adresy **Z** nahrán do registru **R2**.

Jako třetí příklad můžeme uvést instrukci **LPM R2,Z+**. Proti předchozí variantě se po nahrání údaje ještě zvýší obsah registru **Z**, takže tento registr

Tab 13.3. Instrukce přesunů dat (1. část)

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
MOV	Rd,Rr	přesun mezi registry	Rd←Rr	–	1/1
LDI	Rd,data8	nahrání přímé hodnoty, **d je v rozsahu 16 až 31**	Rd←data8	–	1/1
LD	Rd,X	nahrání dat adresovaných nepřímo přes X	Rd←[X]	–	1/2
LD	Rd,X+	nahrání dat adresovaných nepřímo přes X, post-inkrement	Rd←[X], X←X+1	–	1/2
LD	Rd,-X	nahrání dat adresovaných nepřímo přes X, pre-dekrement	X←X-1, Rd←[X]	–	1/2
LD	Rd,Y	nahrání dat adresovaných nepřímo přes Y	Rd←[Y]	–	1/2
LD	Rd,Y+	nahrání dat adresovaných nepřímo přes Y, post-inkrement	Rd←[Y], Y←Y+1	–	1/2
LD	Rd,-Y	nahrání dat adresovaných nepřímo přes Y, pre-dekrement	Y←Y-1, Rd←[Y]	–	1/2
LDD	Rd,Y+off6	nahrání dat adresovaných nepřímo přes Y a posunutí	Rd←[Y+off6]	–	1/2
LD	Rd,Z	nahrání dat adresovaných nepřímo přes Z	Rd←[Z]	–	1/2
LD	Rd,Z+	nahrání dat adresovaných nepřímo přes Z, post-inkrement	Rd←[Z], Z←Z+1	–	1/2
LD	Rd,-Z	nahrání dat adresovaných nepřímo přes Z, pre-dekrement	Z←Z-1, Rd←[Z]	–	1/2
LDD	Rd,Z+off6	nahrání dat adresovaných nepřímo přes Z a posunutí	Rd←[Z+off6]	–	1/2
LDS	Rd,addr16	nahrání přímo adresovaných dat	Rd←[addr16]	–	2/2

Tab 13.3. Instrukce přesunů dat (2. část)

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
ST	X,Rr	uložení dat adresovaných nepřímo přes X	[X]←Rr	–	1/2
ST	X+,Rr	uložení dat adresovaných nepřímo přes X, post-inkrement	[X]←Rr, X←X+1	–	1/2
ST	-X,Rr	uložení dat adresovaných nepřímo přes X, pre-dekrement	X←X-1, [X]←Rr	–	1/2
ST	Y,Rr	uložení dat adresovaných nepřímo přes Y	[Y]←Rr	–	1/2
ST	Y+,Rr	uložení dat adresovaných nepřímo přes Y, post-inkrement	[Y]←Rr, Y←Y+1	–	1/2
ST	-Y,Rr	uložení dat adresovaných nepřímo přes Y, pre-dekrement	Y←Y-1, [Y]←Rr	–	1/2
STD	Y+off6,Rr	uložení dat adresovaných nepřímo přes Y a posunutí	[Y+off6]←Rr	–	1/2
ST	Z,Rr	uložení dat adresovaných nepřímo přes Z	[Z]←Rr	–	1/2
ST	Z+,Rr	uložení dat adresovaných nepřímo přes Z, post-inkrement	[Z]←Rr, Z←Z+1	–	1/2
ST	-Z,Rr	uložení dat adresovaných nepřímo přes Z, pre-dekrement	Z←Z-1, [Z]←Rr	–	1/2
STD	Z+off6,Rr	uložení dat adresovaných nepřímo přes Z a posunutí	[Z+off6]←Rr	–	1/2
STS	addr16,Rr	uložení přímo adresovaných dat	[addr16]←Rd	–	2/2
LPM		nahrání konstanty z paměti programu	R0←[Flash:Z]	–	1/3
LPM	Rd,Z	nahrání konstanty z paměti programu	Rd←[Flash:Z]	–	1/3
LPM	Rd,Z+	nahrání konstanty z paměti programu, zvýšení ukazatele	Rd←[Flash:Z] Z←Z+1	–	1/3
IN	Rd,P	čtení z portu	Rd←P	–	1/1
OUT	P,Rr	zápis na port	P←Rr	–	1/1
PUSH	Rr	uložení do zásobníku	STACK←Rr SP←SP-1	–	1/2
POP	Rd	vyzvednutí ze zásobníku	SP←SP+1 Rd←STACK	–	1/2

ukazuje na následující adresu v programové paměti.

• Velmi důležité jsou instrukce **IN** a **OUT**. Ty pracují se vstupně/výstupními registry mikrořadiče. Přesouvají data mezi vstupně/výstupním registrem a registrem registrového pole. **P** je vstupně/výstupní registr (port), **Rr** je hodnota pro zápis, do **Rd** je uložena přečtená hodnota.

Např. instrukce **IN R0,PIND** přečte stav vstupních vývodů portu D (tento údaj poskytuje v/v registr **PIND**) a uloží jej do registru **R0**.

Instrukce **OUT PORTB,R16** zase odešle obsah registru **R16** na port B (výstupní buffer tohoto portu odpovídá v/v registru **PORTB**).

• Instrukce **PUSH** a **POP** slouží pro ukládání/výběr hodnot do/ze zásobníku. Práce se zásobníkem byla vysvětlena v kapitole 2.4 (v KE 5/2006).

Při vložení údaje (**PUSH Rr**) se hodnota **Rr** vloží na vrchol zásobníku a ukazatel vrcholu zásobníku (registr **SP**) sníží o jedničku.

Při vybírání údaje (**POP Rd**) je režie opačná. Ukazatel vrcholu zásobníku se nejdříve přesune na další adresu a z takto stanoveného vrcholu se vyzvedne hodnota do registru **Rd**.

Časování instrukcí přesunů je závislé na tom, zda se pracuje s registry nebo s pamětí. Instrukce pracující výhradně nad registry trvají pouze jeden hodinový cyklus. Práce s datovou pamětí je 2cyklová a práce s programovou pamětí pak 3cyklová. Délka instrukcí je jedno slovo, výjimkou jsou instrukce **LDS** a **STS**, které mají délku dvě slova.

13.5. Bitové operace

Bitové operace (tab. 13.4) pracují nad příznakem registru **SREG** nebo nad bitem registru registrového pole nebo vstupně/výstupního registru určeným jeho číslem. Dále sem patří instrukce posuvů a rotací.

• Instrukce **SBI** a **CBI** nastavují/nulují bit **b** (0 až 7) vstupně/výstupního registru (portu) **P**. Důležité je připomenout, že **P** je v rozsahu 0 až 31. Znamená to, že takto lze ovládat pouze dolní polovinu vstupně/výstupních registrů (6bitové číslo portu, které by adresovalo všechny porty, by se „nevešlo" do strojového kódu instrukce) - viz tab. 12.1.

Např. instrukce **SBI DDRB,0** nastaví nejnižší bit portu B do výstupního směru. Instrukce **CBI PORTB,0** zase přivede na nejnižší bit portu B (PORTB0) úroveň „log. 0".

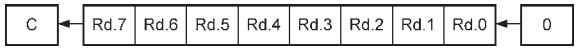
• Instrukce **LSL** a **LSR** provádějí logický posuv registru **Rd** o jedno místo (bit) doleva/doprava (obr. 13.1 a 13.2). Logickým posuvem se myslí, že vsouvá nula zprava/zleva. Bit, který „vypadne", je uložen do příznaku C.

• Instrukce **ROL** a **ROR** provádějí rotaci registru **Rd** o jedno místo (bit) doleva/doprava (obr. 13.3 a 13.4). Při této operaci se registr **Rd** spojí s příznakem **C** do 9bitového registru a všechny bity se posunou o jedno místo doleva/doprava.

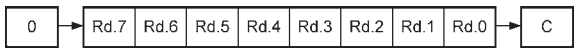
Tab 13.4. Bitové operace

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
SBI	P,b	nastaví bit **b** v/v registru **P**, **P je v rozsahu 0 až 31**	P.b←1	–	1/2
CBI	P,b	nuluje bit **b** v/v registru **P**, **P je v rozsahu 0 až 31**	P.b←0	–	1/2
LSL	Rd	logický posuv registru **Rd** doleva	obr. 13.1	Z,C,N,V	1/1
LSR	Rd	logický posuv registru **Rd** doprava	obr. 13.2	Z,C,N,V	1/1
ROL	Rd	rotace registru **Rd** doleva přes **C**	obr. 13.3	Z,C,N,V	1/1
ROR	Rd	rotace registru **Rd** doprava přes **C**	obr. 13.4	Z,C,N,V	1/1
ASR	Rd	aritmetický posuv registru **Rd** doprava	obr. 13.5	Z,C,N,V	1/1
SWAP	Rd	výměna dolní a horní čtveřice **Rd**	low(Rd) ←high(Rd) high(Rd) ←low(Rd)	–	1/1
BSET	s	nastavení příznaku **s** registru **SREG**	SREG.s←1	SREG.s	1/1
BCLR	s	nulování příznaku **s** registru **SREG**	SREG.s←0	SREG.s	1/1
BST	Rr,b	přenes bit **b** registru **Rr** do příznaku **T**	T←Rr.b	T	1/1
BLD	Rd,b	přenes příznak **T** do bitu **b** registru **Rd**	Rd.b←T	–	1/1
SEC		nastavení příznaku **C**	C←1	C	1/1
CLC		nulování příznaku **C**	C←0	C	1/1
SEN		nastavení příznaku **N**	N←1	N	1/1
CLN		nulování příznaku **N**	N←0	N	1/1
SEZ		nastavení příznaku **Z**	Z←1	Z	1/1
CLZ		nulování příznaku **Z**	Z←0	Z	1/1
SEI		nastavení příznaku **I**, povolení přerušení	I←1	I	1/1
CLI		nulování příznaku **I**, zákaz přerušení	I←0	I	1/1
SES		nastavení příznaku **S**	S←1	S	1/1
CLS		nulování příznaku **S**	S←0	S	1/1
SEV		nastavení příznaku **V**	V←1	V	1/1
CLV		nulování příznaku **V**	V←0	V	1/1
SET		nastavení příznaku **T**	T←1	T	1/1
CLT		nulování příznaku **T**	T←0	T	1/1
SEH		nastavení příznaku **H**	H←1	H	1/1
CLH		nulování příznaku **H**	H←0	H	1/1

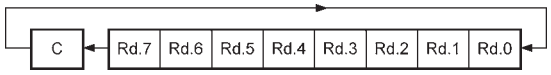
Obr. 13.1. Výklad instrukce LSL



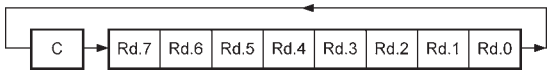
Obr. 13.2. Výklad instrukce LSR



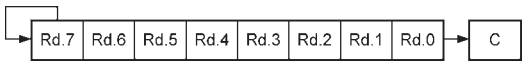
Obr. 13.3. Výklad instrukce ROL



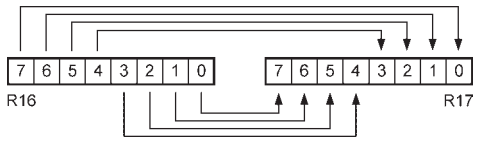
Obr. 13.4. Výklad instrukce ROR



Obr. 13.5. Výklad instrukce ASR



Obr. 13.6. Vysvětlení operace „přetočení" bitů



• Instrukce **ASR Rd** provádí tzv. aritmetický posuv registru **Rd** doprava (obr. 13.5). Připomeňme, že logický a aritmetický posuv doleva jsou stejné, takže pro realizaci aritmetického posuvu doleva se použije instrukce **LSL Rd**.

Při aritmetickém posuvu doprava se bity posunou o jedno místo doprava, nejnižší bit se vloží do příznaku **C**. Na místo uvolněného nejvyššího bitu se vloží předchozí hodnota. Takže aritmetický posuv doprava čísla 01001101_2 je 00100110_2, ale aritmetický posuv doprava čísla 11001101_2 je 11100110_2. Je to kvůli druhému doplňku (změna nejvyššího bitu totiž vede ke znaménku čísla, které je zapsáno ve druhém doplňku).

Vzhledem k tomu, že operace logického posuvu doleva odpovídá vlastně násobení 2 a operace posuvu doprava pak dělení 2, lze tyto instrukce používat pro dělení a násobení.

Jako praktický příklad si předvedeme, jak lze číslo obsažené v registru **R16** vynásobit 10. Jelikož násobit lze pouze mocninami základu 2, musíme si číslo 10 rozložit jako 3 posuny doleva (násobení 8) a jeden posun doleva (násobení 2). Posuneme-li původní obsah registru **R16** třikrát doleva, vynásobíme jej osmi. Pokud původní hodnotu ještě posuneme doleva jednou (tedy vynásobíme dvěma) a sečteme s předchozím mezivýsledkem, budeme mít opravdu k dispozici desetinásobek původní hodnoty:

```
MOV R17,R16       ;uchovej výchozí obsah R16
LSL R16           ;x2
LSL R16           ;x2
LSL R16           ;x2, celkem násobeno osmi
LSL R17           ;x2
ADD R16,R17       ;soucet mezivýsledků do R16
```

• Instrukce **SWAP Rd** provede výměnu horní a dolní čtveřice bitů registru **Rd**. Tato operace je výhodná například při rozkladech 8bitových čísel na dvě hexadecimální číslice.

• Instrukce **BSET** a **BCLR** nastavují/nulují zvolený bit **s** stavového registru **SREG**. Jinou možností je použít instrukce **SEC**, **CLC**, **SEN**, **CLN**, **SEZ**, **CLZ**, **SEI**, **CLI**, **SES**, **CLS**, **SEV**, **CLV**, **SET**, **CLT**, **SEH** a **CLH**.

• Instrukce **BST** a **BLD** slouží pro přenos bitu **b** z registru **Rr**/příznaku **T** do příznaku **T**/registru **Rr**. Instrukce **BST** se obvykle použije pro rozklad čísla obsaženého v registru **Rr** na jednotlivé bity. Podobně instrukcí **BLD** lze bity postupně složit do podoby čísla, které se objeví v registru **Rd**.

Pěkným příkladem použití instrukcí **BST** a **BLD** je „přetočení" pořadí bitů (obr. 13.6). V registru **R16** máme hodnotu, kterou chceme zkopírovat do registru **R17** v opačném pořadí bitů. Například nejvyšší bit **R16** se má nahrát do nejnižšího bitu **R17**.

V první fázi musíme přečíst hodnotu jednoho bitu registru **R16** a potom ji přenést do určeného bitu registru **R17**. Tuto operaci opakujeme i s ostatními bity:

```
BST R16,7         ;nacti bit 7
BLD R17,0         ;ulož jako bit 0
BST R16,6         ;nacti bit 6
BLD R17,1         ;ulož jako bit 1
BST R16,5         ;nacti bit 5
BLD R17,2         ;ulož jako bit 2
BST R16,4         ;nacti bit 4
BLD R17,3         ;ulož jako bit 3
BST R16,3         ;nacti bit 3
BLD R17,4         ;ulož jako bit 4
BST R16,2         ;nacti bit 2
BLD R17,5         ;ulož jako bit 5
BST R16,1         ;nacti bit 1
BLD R17,6         ;ulož jako bit 6
BST R16,0         ;nacti bit 0
BLD R17,7         ;ulož jako bit 7
```

13.6. Skoky a přeskoky

Instrukce uvedené v tab. 13.5 až tab. 13.7 se používají pro změnu pro-

Tab 13.5. Instrukce skoků

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
RJMP	rel12	nepodmíněný relativní skok	PC←PC+rel12+1	–	1/2
IJMP		nepodmíněný absolutní nepřímý skok	PC←[Z]	–	1/2
BRBS	s,rel7	provede rel. skok, je-li SREG.s = 1	je-li SREG.s=1, PC←PC+rel7+1	–	1/1 až 2
BRBC	s,rel7	provede rel. skok, je-li SREG.s = 0	je-li SREG.s=0, PC←PC+rel7+1	–	1/1 až 2
BREQ	rel7	provede rel. skok, je-li Z = 1 (výsledek je shoda)	je-li Z=1, PC←PC+rel7+1	–	1/1 až 2
BRNE	rel7	provede rel. skok, je-li Z = 0 (výsledek je rozdíl)	je-li Z=0, PC←PC+rel7+1	–	1/1 až 2
BRCS	rel7	provede rel. skok, je-li C = 1	je-li C=1, PC←PC+rel7+1	–	1/1 až 2
BRCC	rel7	provede rel. skok, je-li C = 0	je-li C=0, PC←PC+rel7+1	–	1/1 až 2
BRSH	rel7	provede rel. skok, je-li C = 1 (větší, rovno)	je-li C=0, PC←PC+rel7+1	–	1/1 až 2
BRLO	rel7	provede rel. skok, je-li C = 0 (menší)	je-li C=1, PC←PC+rel7+1	–	1/1 až 2
BRMI	rel7	provede rel. skok, je-li N = 1 (záporné)	je-li N=1, PC←PC+rel7+1	–	1/1 až 2
BRPL	rel7	provede rel. skok, je-li N = 0 (kladné)	je-li N=0, PC←PC+rel7+1	–	1/1 až 2
BRGE	rel7	provede rel. skok, je-li větší rovno (znaménkově)	je-li N=1, PC←PC+rel7+1	–	1/1 až 2
BRLT	rel7	provede rel. skok, je-li menší než 0 (znaménkově)	je-li N=0, PC←PC+rel7+1	–	1/1 až 2
BRHS	rel7	provede rel. skok, je-li H = 1	je-li H=1, PC←PC+rel7+1	–	1/1 až 2
BRHC	rel7	provede rel. skok, je-li H = 0	je-li H=0, PC←PC+rel7+1	–	1/1 až 2
BRTS	rel7	provede rel. skok, je-li T = 1	je-li T=1, PC←PC+rel7+1	–	1/1 až 2
BRTC	rel7	provede rel. skok, je-li T = 0	je-li T=0, PC←PC+rel7+1	–	1/1 až 2
BRVS	rel7	provede rel. skok, je-li V = 1	je-li V=1, PC←PC+rel7+1	–	1/1 až 2
BRVC	rel7	provede rel. skok, je-li V = 0	je-li V=0, PC←PC+rel7+1	–	1/1 až 2
BRIE	rel7	provede rel. skok, je-li povoleno přerušení	je-li I=1, PC←PC+rel7+1	–	1/1 až 2
BRID	rel7	provede rel. skok, je-li zakázáno přerušení	je-li I=0, PC←PC+rel7+1	–	1/1 až 2

Tab 13.6. Přehled podmíněných skoků

Čísla bez znaménka					
Test	**Log. výraz**	**Instrukce**	**Opačný test**	**Log. výraz**	**Instrukce**
Rd=Rr	Z=1	BREQ	Rd≠Rr	Z=0	BRNE
Rd<Rr	C=1	BRLO/BRCS	Rd≥Rr	C=0	BRSH/BRCC
Rd≤Rr	C or Z=1	BRSH*	Rd>Rr	C or Z=0	BRLO*
Rd>Rr	C or Z=0	BRLO*	Rd≤Rr	C or Z=1	BRSH*
Rd≥Rr	C=0	BRSH/BRCC	Rd<Rr	C=1	BRLO/BRCC

Čísla se znaménkem					
Test	**Log. výraz**	**Instrukce**	**Opačný test**	**Log. výraz**	**Instrukce**
Rd=Rr	Z=1	BREQ	Rd≠Rr	Z=0	BRNE
Rd<Rr	(N xor V)=1	BRLT	Rd≥Rr	(N xor V)=0	BRGE
Rd≤Rr	Z or (N xor V)=1	BRGE*	Rd>Rr	Z and (N xor V)=0	BRLT*
Rd>Rr	Z and (N xor V)=1	BRLT*	Rd≤Rr	Z or (N xor V)=0	BRGE*
Rd≥Rr	(N xor V)=0	BRGE	Rd<Rr	(N xor V)=1	BRLT

Jednoduché testy					
Test	**Log. výraz**	**Instrukce**	**Opačný test**	**Log. výraz**	**Instrukce**
C	C=1	BRCS	$\overline{C}$	C=0	BRCC
N	N=1	BRMI	$\overline{N}$	N=0	BRPL
V	V=1	BRVS	$\overline{V}$	V=0	BRVC
Z	Z=1	BREQ	$\overline{Z}$	Z=0	BRNE

* značí, že se operandy (registry Rd a Rr) musí vzájemně vyměnit

Tab 13.7. Instrukce přeskoků

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
CPSE	Rd,Rr	přeskoč při shodě (**Rd = Rr**)	je-li Rd=Rr, PC←PC+2/3	–	1/1 až 3
SBRC	Rr,b	přeskoč, je-li bit **b** registru **Rr** vynulován	je-li Rr.b=0, PC←PC+2/3	–	1/1 až 3
SBRS	Rr,b	přeskoč, je-li bit **b** registru **Rr** nastaven	je-li Rr.b=1, PC←PC+2/3	–	1/1 až 3
SBIC	P,b	přeskoč, je-li bit **b** portu **P** vynulován, **P je v rozsahu 0 až 31**	je-li P.b=0, PC←PC+2/3	–	1/1 až 3
SBIS	P,b	přeskoč, je-li bit **b** portu **P** nastaven, **P je v rozsahu 0 až 31**	je-li P.b=1, PC←PC+2/3	–	1/1 až 3

gramového řízení (pro přechod do jiné části programu) a dělí se do tří skupin:

- **nepodmíněné skoky (jumps)** - provedou se vždy (jejich provedení není podmíněno splněním nějaké podmínky),
- **podmíněné skoky (branchs)** - provedou se jen v tom případě, že je splněna určitá podmínka (např. je nastaven příznak C), jinak se pokračuje následující instrukcí,
- **přeskoky (skips)** - při splnění určité podmínky přeskočí následující instrukci, jinak se pokračuje následující instrukcí.

Instrukce **RJMP rel12** je relativním nepodmíněným skokem. Provede se vždy (nemusí být splněna žádná podmínka), dosah skoku je -2048 až +2047 pozic vůči aktuálnímu místu programu (což je PC + 1).

Instrukce **IJMP** je nepodmíněným nepřímým skokem v absolutním rozsahu adres 0 až 65535. Instrukce nemá operandy, místo skoku je určeno obsahem registru **Z**.

Následují **instrukce podmíněných skoků**. Tyto skoky jsou relativní 7bitové, dosah skoku je -64 až +63 pozic vůči aktuálnímu místu programu (PC). Časová náročnost instrukce závisí na tom, zda se skok zrealizuje (zda je podmínka splněna) nebo ne. Pokud je podmínka splněna, trvá instrukce 2 cykly. Pokud ne, trvá jeden cyklus.

Podmíněné skoky se provedou pouze v případě splnění určité podmínky. Situace je poměrně složitá, proto je uvedena tabulka 13.6, pomocí které si zlepšíme přehled. V tabulce jsou uvedeny instrukce porovnávající čísla bez znaménka, čísla se znaménkem a jednoduché testy příznaků z registru **SREG**.

Velmi zajímavou skupinou instrukcí jsou tzv. přeskoky. Jedná se o skoky, které při splnění podmínky přeskočí instrukci následující za přeskokem. Pokud podmínka splněna není, je následující instrukce vykonána.

Časové provedení přeskoku je závislé jednak na splnění podmínky, ale také na délce následující instrukce. Pokud není podmínka splněna, trvá zpracování přeskoku jeden hodinový cyklus. Při splnění podmínky (přeskočení následující instrukce), trvá přeskok 2 nebo 3 hodinové cykly (zavisí totiž na tom, zda je následující instrukce 1cyklová nebo 2cyklová).

Instrukční soubor mikrořadičů AVR obsahuje 5 přeskoků:

- **CPSE Rd,Rr** porovnává obsahy registrů **Rd** a **Rr**. Jsou-li jejich hodnoty stejné, přeskočí se následující instrukce,
- **SBRC** a **SBRS** testují určený bit (0/1) registru **Rr** registrového pole,
- **SBIC** a **SBIS** testují určený bit (0/1) vstupně/výstupního registru **P** (port musí být v rozsahu 0 až 31).

Jako praktický příklad si předvedeme smyčku čekající na přivedení úrovně „log. 1“ na vývod **PB7**.

```
CEKEJ:  SBIS PINB,7      ;testuj PB7
        RJMP CEKEJ       ;PB7=0, zpet
             ...         ;další instrukce
```

Pokud je **PB7 = 0** není přeskok **SBIS PINB,7** proveden a instrukce **RJMP CEKEJ** vrací řízení programu na návěští **CEKEJ**. V opačném případě (PB7 = 1) zajistí přeskok **SBIS PINB,7** přechod na další instrukce.

13.7. Instrukce pro podporu podprogramů

Mnohdy je třeba několikrát opakovat určité sekvence instrukcí. Jistě není účelné je zapisovat znovu a znovu až do omrzení. Také by narostla délka cílového kódu. Proto je vhodné používat **podprogramy**.

Podprogram se od ostatních řádků programu příliš neliší. Pouze je třeba jej zavolat (pomocí **RCALL** nebo **ICALL**) a potom se z něj vrátit zpět (pomocí **RET**) - viz tab. 13.8.

- Instrukce **RCALL** nebo **ICALL** se chovají podobně jako instrukce skoků **RJMP** nebo **IJMP**. Aby se však podprogram mohl vrátit zpět do místa, odkud byl zavolán, musí se nejdříve uložit tzv. návratová adresa do zásobníku. Návratová adresa je vlastně adresa instrukce následující za **RCALL** nebo **ICALL** (tuto adresu obsahuje **PC**). Podobně jako u instrukce **PUSH** je nejdříve **SP** snížen (nyní o 2) a potom se do zásobníku uloží 16bitová návratová adresa.
- Instrukce **RET** slouží pro návrat z podprogramu. Režie spojená s vykonáním této instrukce je nepatrná, protože vše

Tab 13.8. Instrukce pro podporu podprogramů

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
RCALL	rel12	relativní volání podprogramu	STACK←PC+1 SP←SP-2 PC←PC+rel12+1	–	1/3
ICALL		nepřímé volání podprogramu	STACK←PC+1 SP←SP-2 PC←[Z]	–	1/3
RET		návrat z podprogramu	PC←STACK SP←SP+2	–	1/4
RETI		návrat z rutiny obsluhy přerušení	PC←STACK SP←SP+2	I = 1	1/4

Tab 13.9. Logické operace

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
AND	Rd,Rr	logický součin registrů **Rd** a **Rr**	Rd←Rd and Rr	Z,N,V	1/1
ANDI	Rd,data8	logický součin registru **Rd** a konstanty, **d je v rozsahu 16 až 31**	Rd←Rd and data8	Z,N,V	1/1
OR	Rd,Rr	logický součet registrů **Rd** a **Rr**	Rd←Rd or Rr	Z,N,V	1/1
ORI	Rd,data8	logický součet registru **Rd** a konstanty, **d je v rozsahu 16 až 31**	Rd←Rd or data8	Z,N,V	1/1
EOR	Rd,Rr	výlučný logický součet registrů **Rd** a **Rr**	Rd←Rd xor Rr	Z,N,V	1/1
COM	Rd	negace (1. doplněk) registru **Rd**	Rd←$FF-Rd	Z,C,N,V	1/1
SBR	Rd,data8	nastaví bity registru **Rd** podle konstanty, **d je v rozsahu 16 až 31**	Rd←Rd or data8	Z,N,V	1/1
CBR	Rd,data8	nuluje bity registru **Rd** podle konstanty, **d je v rozsahu 16 až 31**	Rd←Rd and (/data8*)	Z,N,V	1/1
TST	Rd	test nuly nebo záporného čísla v **Rd**	Rd←Rd and Rd	Z,N,V	1/1
CLR	Rd	vynulování registru **Rd**	Rd←Rd xor Rd	Z,N,V	1/1
SER	Rd	nastavení všech bitů registru **Rd**	Rd←$FF	–	1/1

*/data8 značí negaci data8

připravila již instrukce **RCALL** resp. **ICALL**. Ze zásobníku se vyzvednou 2 bajty a nastaví se do **PC**. Zároveň se **SP** zvýší o 2.

Pozor na správnou manipulaci se zásobníkem! Pokud si do něj na začátku podprogramu uložíte nějakou další hodnotu (kromě návratové adresy) a zapomenete ji vyjmout před použitím instrukce **RET**, nebude použita správná návratová adresa (skočí se úplně jinam)!

• Instrukce **RETI** slouží pro návrat z tzv. rutiny obsluhy přerušení. Rutina obsluhy přerušení je vlastně také podprogram, který vyvolá sám procesor, pokud dojde k přerušení. Procesor v tomto případě automaticky zakáže všechna přerušení. Takže jedinou věcí, kterou musí **RETI** zajistit navíc oproti **RET**, je povolit přerušení.

13.8. Logické operace

Logickými operacemi jsou logický součin, součet, výlučný součet a negace (sestavení prvního doplňku), nastavení a nulování bitů (tab. 13.9).

• Instrukce **AND/ANDI** představuje logický součin dvou operandů. Výsledek míří do levého operandu. Při logickém součinu jsou oba operandy brány jako jednotlivé bity, takže logický součin proběhne pro všech 8 bitů najednou. Logický součin se nejčastěji používá pro vymaskování jednoho bitu nebo celé skupiny bitů. To znamená, že některé bity chceme zamaskovat, tedy vlastně vynulovat (také lze použít instrukci **CBR**).

• Instrukce **OR/ORI** představuje logický součet dvou operandů. Výsledek míří do levého operandu. Při logickém součtu jsou oba operandy brány jako jednotlivé bity, takže logický součet proběhne pro všech 8 bitů najednou. Logický součet se nejčastěji používá pro nastavení jednoho bitu nebo celé skupiny bitů (také lze použít instrukci **SBR**).

• Instrukce **EOR** představuje výlučný logický součet dvou operandů. Výsledek míří do levého operandu. Při výlučném logickém součtu jsou oba operandy brány jako jednotlivé bity, takže výlučný logický součet proběhne pro všech 8 bitů najednou. Výlučný logický součet se nejčastěji používá pro negování jednoho bitu nebo celé skupiny bitů.

Obr. 13.7 ilustruje použití instrukcí **AND**, **OR** a **EOR**. Nejdříve je ukázáno vymaskování dolních 4 bitů pomocí **AND**. Druhý příklad ukazuje nastavení bitů s pořadovými čísly 5 a 2 pomocí **OR**. Třetí příklad ukazuje negaci nejnižšího bitu pomocí **EOR**.

• Instrukce **COM** provede negaci všech bitů zvoleného registru registrového pole. Všechny bity tohoto registru tedy změní své hodnoty na opačné. Tato operace odpovídá tzv. prvnímu doplňku.

• Instrukce **SBR** a **CBR** slouží k nastavení nebo nulování několika bitů zvoleného registru registrového pole najednou. Bit, který je v pravém operandu (konstantě) nastaven, bude ve výsledku nastaven (**SBR**) nebo vynulován (**CBR**). Např. $03 nastaví/vynuluje dolní dva bity.

• Instrukce **TST** zaktualizuje příznaky **Z** a **N** tak, aby bylo možné otestovat hodnotu v registru **Rd**. Pak se tedy dá testovat nulovost nebo znaménko čísla v registru **Rd** (připomeňme, že tyto příznaky se obvykle zaktualizují automaticky po provedení aritmetické operace).

```
    10111010      11100001      11101101
and 00001111  or  00100100  xor 00000001
    --------      --------      --------
    00001010      11100101      11101110
```

Obr. 13.7. Výklad instrukcí AND, OR a EOR

• Instrukce **CLR** a **SER** zajišťují rychlé vynulování nebo nastavení všech bitů registru **Rd**. To je velmi častá operace (není třeba používat instrukci **LDI**, která pracuje pouze nad horní polovinou registrového pole).

13.9. Aritmetické operace

Aritmetické instrukce umožňují provádění celočíselného součtu, rozdílu a dalších operací. Při aritmetických operacích se mění velký počet příznaků.

• Instrukce **ADD Rd,Rr** je celočíselný součet dvou registrů registrového pole, výsledek je uložen do levého registru (**Rd**). Při výpočtu může dojít ke změně všech příznaků:

a) **Z** bude nastaven, je-li výsledek součtu nula,

b) **C** bude nastaven, pokud se bude generovat přenos ze sedmého bitu (výsledek přeteče a „nevejde" se do **Rd**). V opačném případě je **C** vynulován.

c) **H** bude nastaven, pokud se bude generovat přenos ze třetího bitu (přenos mezi dolní a horní čtveřicí bitů výsledku). V opačném případě je **H** vynulován.

d) **N** je nastaven, je-li výsledek záporný (to nastane, když je nejvyšší bit výsledku nastaven). Tento bit má význam pouze při práci s čísly se znaménkem!

e) **V** bude nastaven, pokud se přeplní rozsah zobrazení čísel v druhém doplňku (čísel se znaménkem). To se projeví změnou znaménka výsledku. Pokud nepracujeme s čísly se znaménkem, nemá **V** význam!

*V předchozím textu byl použit pojem **číslo se znaménkem**. Na tomto místě je nutné podat vysvětlení:*

V registrech mohou být uložena čísla, která bereme tak, že jsou bez znaménka (rozsah je 0 až 255) nebo se znaménkem (rozsah je -128 až +127) - viz obr. 13.8.

Zobrazení čísel bez znaménka je prosté, hodnoty se postupně zvyšují od samých nul až po samé jedničky.

Tab 13.10. Aritmetické operace

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
ADD	Rd,Rr	sečte registry **Rd** a **Rr**	Rd←Rd+Rr	Z,C,N,V,H	1/1
ADC	Rd,Rr	sečte registry **Rd** a **Rr** a příznak **C**	Rd←Rd+Rr+C	Z,C,N,V,H	1/1
ADIW	Rd,data6	přičte konstantu ke slovu **Rd+1:Rd** **d jsou hodnoty 24, 26, 28, 30**	Rd+1:Rd← Rd+1:Rd +data6	Z,C,N,V,S	1/2
SUB	Rd,Rr	odečte registry **Rd** a **Rr**	Rd←Rd-Rr	Z,C,N,V,H	1/1
SUBI	Rd,data8	odečte od registru **Rd** konstantu, **d je v rozsahu 16 až 31**	Rd←Rd-data8	Z,C,N,V,H	1/1
SBIW	Rd,data6	odečte konstantu od slova **Rd+1:Rd** **d jsou hodnoty 24, 26, 28, 30**	Rd+1:Rd← Rd+1:Rd -data6	Z,C,N,V,S	1/2
SBC	Rd,Rr	odečte registry **Rd** a **Rr** a příznak **C**	Rd←Rd-Rr-C	Z,C,N,V,H	1/1
SBCI	Rd,data8	odečte od registru **Rd** konstantu a **C**, **d je v rozsahu 16 až 31**	Rd←Rd -data8-C	Z,C,N,V,H	1/1
INC	Rd	inkrementuje registr **Rd**	Rd←Rd+1	Z,N,V	1/1
DEC	Rd	dekrementuje registr **Rd**	Rd←Rd-1	Z,N,V	1/1
NEG	Rd	druhý doplněk registru **Rd**	Rd←$00-Rd	Z,C,N,V	1/1

NORMÁLNÍ ZOBRAZENÍ		DRUHÝ DOPLŇEK	
dvojkově	desítkově	dvojkově	desítkově
11111111	255	01111111	+127
11111110	254	01111110	+126
		01111101	+125
...		...	
...		00000001	+1
...		00000000	0
...		11111111	–1
...		...	
...		10000010	–126
00000001	1	10000001	–127
00000000	0	10000000	–128

Obr. 13.8. Zobrazení čísel 0 až 255 a -128 až +127

*Čísla se znaménkem se zobrazují v tzv. **druhém doplňku**. Kladné hodnoty včetně nuly jsou zobrazeny stejně, jako odpovídající čísla bez znamének (00000000 až 01111111), což odpovídá rozsahu 0 až +127. Záporná čísla (-128 až -1) jsou zobrazena jako 10000000 až 11111111. Druhý doplněk (počítá jej instrukce **NEG**) se stanoví jako $FF-hodnota (jiný způsob je negace všech bitů a přičtení 1).*

Důvod takto „podivného" zobrazení záporných čísel spočívá v aritmetice. Všechny záporné hodnoty mají nejvyšší bit nastaven, takže se snadno rozliší kladné a záporné hodnoty. Procesor bude navíc počítat s takto zapsanou hodnotou korektně. Např. výraz 12 - 10 = 2 nebo 10 - 12 = -2.

Na obr. 13.9 jsou uvedeny příklady několika výpočtů součtů. Součet je zobrazen binárně (tak pracuje procesor), desítkově z pohledu čísel bez znaménka a desítkově z pohledu čísel se znaménkem.

• Instrukce **ADC Rd,Rr** má podobnou funkci jako **ADD**. V tomto případě je obsah registru **Rd** sčítán s registrem **Rr** a s příznakem **C**.

Instrukce je vhodná např. na sčítání čísel delších než 8 bitů. Níže uvedený kód sečte dvě 16bitová čísla (první číslo je v registrech **R1** - vyšší bajt a **R0** - nižší bajt; druhé číslo je v registrech **R3** - vyšší bajt a **R2** - nižší bajt; výsledek je uložen do **R1** - vyšší bajt a **R0** - nižší bajt):

```
ADD R0,R2     ;secti dolní bajty
ADC R1,R3     ;secti horní bajty+C
```

• Instrukce **ADIW** přičítá 6bitovou konstantu (0 až 63) ke slovu tvořenému párem registrů **R25:R24**, **R27:R26**, **R29:R28** nebo **R31:R30**. Tato instrukce je tedy speciálně použitelná pro posuv ukazatelových registrů **X**, **Y**, **Z** (připomeňme, že např. registr Z odpovídá R31:R30). Takto lze tedy snadno realizovat posunutí ukazatele o relativní vzdálenost až +63 bajtů.

• Instrukce **SUB Rd,Rr** odečte dva registry registrového pole od sebe. Výsledek je uložen do levého registru (**Rd**). Mohou se změnit příznaky:

a) **Z** bude nastaven, je-li výsledek rozdílu nula,

b) **C** bude nastaven, pokud bude požadována výpůjčka ze sedmého bitu (výsledek podteče a nevejde se do registru). V opačném případě je **C** vynulován.

c) **H** bude nastaven, pokud bude požadována výpůjčka ze třetího bitu (výpůjčka mezi dolní a horní čtveřicí bitů). V opačném případě je **H** vynulován.

d) **N** je nastaven, je-li výsledek záporný (to nastane, když je nejvyšší bit výsledku nastaven). Tento bit má význam pouze při práci s čísly se znaménkem!

e) **V** bude nastaven, pokud se přeplní rozsah zobrazení čísel v druhém doplňku (čísel se znaménkem). To se projeví změnou znaménka výsledku. Pokud nepracujeme s čísly se znaménkem, nemá **V** význam!

Na obr. 13.10 jsou příklady několika výpočtů rozdílů. Rozdíl je zobrazen binárně (tak pracuje procesor), desítkově z pohledu čísel bez znaménka a desítkově z pohledu čísel se znaménkem.

• Instrukce **SUBI Rd,Rr** je podobná instrukci **SUB**. Pravý operand rozdílu je nyní představován konstantou (tedy ne obsahem registru). Číslo cílového registru **Rd** je v rozsahu 16 až 31!

• Instrukce **SBIW** odečítá 6bitovou konstantu (0 až 63) od slova tvořeného párem registrů R25:R24, R27:R26, R29:R28 nebo R31:R30. Tato instrukce je tedy speciálně použitelná pro posuv ukazatelových registrů X, Y, Z (připomeňme, že např. registr Z odpovídá R31:R30). Takto lze tedy snadno realizovat posunutí ukazatele o relativní vzdálenost až -63 bajtů.

• Instrukce **SBC** a **SBCI** jsou podobné instrukcím **SUB** a **SUBI**. V rozdílu se navíc objevuje příznak **C** jako výpůjčka.

• Instrukce **INC Rd** a **DEC Rd** slouží k rychlému zvýšení (**INC** jako inkrement) nebo snížení (**DEC** jako dekrement) registru o 1. To je programech častá operace. Pozor, instrukce nemění příznaky **C**, **H**. Pokud dojde k přetečení při této instrukci, nebude zjištěno. Budeme-li testovat **C** nebo **H** po této instrukci, testujeme jejich stav, před provedením této instrukce:

```
INC  R0          ;zvyš obsah R0 (nemení C!!!)
BRCS SKOK        ;skok se provede v závislosti
                 ;na obsahu C pred provedením INC R0
```

• Instrukce **NEG Rd** provádí druhý doplněk registru **Rd**. Druhý doplněk je vlastně stanovení záporné hodnoty čísla se znaménkem. Např. pro +67 vrátí -67, pro -120 vrátí +120.

13.10. Porovnávací instrukce

V souvislosti se skoky se často používají 3 specifické instrukce, které porovnávají obsah registru s jiným registrem nebo konstantou (viz tab. 13.11). Rozvětvení programu pak zajistíme pomocí podmíněných skoků.

• Instrukce **CP Rd,Rr** porovnává registry **Rd** a **Rr** tak, že je odečte od sebe.

Tab 13.11. Porovnávací instrukce

Kód	Operandy	Popis	Operace	Příznaky	Slova /hodiny
CP	Rd,Rr	porovnání registrů **Rd** a **Rr**	Rd-Rr	Z,C,N,V,H	1/1
CPC	Rd,Rr	porovnání registrů **Rd** a **Rr** včetně **C**	Rd-Rr-C	Z,C,N,V,H	1/1
CPI	Rd,data8	porovnání registru **Rd** s konstantou, **d je v rozsahu 16 až 31**	Rd-data8	Z,C,N,V,H	1/1

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
01111111	127	+127	C = 0 (nepřeteklo)
			H = 1 (přenos mezi 3. a 4. bitem)
+ 00000001	+ 1	+ +1	V = 1 (přeplnění)
			N = 1 (záporný výsledek)
10000000	**128**	**–128**	Z = 0 (není nula)

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
10000000	128	–128	C = 1 (přeteklo)
			H = 0 (nebyl přenos mezi 3. a 4. bitem)
+ 11111111	+ 255	+ +1	V = 1 (přeplnění)
			N = 0 (nezáporný výsledek)
01111111	**127**	**+127**	Z = 0 (není nula)

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
00100101	37	+37	C = 0 (nepřeteklo)
			H = 1 (přenos mezi 3. a 4. bitem)
+ 00011101	+ 29	+ +29	V = 0 (nebylo přeplnění)
			N = 0 (nezáporný výsledek)
01000010	**66**	**+66**	Z = 0 (není nula)

Obr. 13.9. Výklad instrukce ADD na příkladech

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
10000000	128	–128	C = 0 (bez výpůjčky)
			H = 1 (výpůjčka mezi 3. a 4. bitem)
– 00000001	– 1	– +1	V = 1 (přeplnění)
			N = 0 (nezáporný výsledek)
01111111	**127**	**+127**	Z = 0 (není nula)

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
01111111	127	+127	C = 1 (výpůjčka)
			H = 0 (nebyla výpůjčka mezi 3. a 4. bitem)
– 11111111	– 255	– –1	V = 1 (přeplnění)
			N = 1 (záporný výsledek)
10000000	**128**	**–128**	Z = 0 (není nula)

dvojkově	desítkově bez znaménka	desítkově se znaménkem	
00100101	37	+37	C = 0 (nebyla výpůjčka)
			H = 1 (výpůjčka mezi 3. a 4. bitem)
– 00011101	– 29	– +29	V = 0 (nebylo přeplnění)
			N = 0 (nezáporný výsledek)
00001000	**8**	**+8**	Z = 0 (není nula)

Obr. 13.10. Výklad instrukce SUB na příkladech

Tab 13.12. Další instrukce

Kód Operandy	Popis	Operace	Příznaky	Slova /hodiny
NOP	prázdná operace, časování	viz popis	–	1/1
SLEEP	zvolí režim spánku	viz popis	–	1/1
WDR	nuluje WDT	viz popis	–	1/1
BREAK	nuluje WDT	viz popis	–	?

Výsledek ale není uložen (což je rozdíl proti instrukci **SUB**). Takže porovnání nezpůsobí změnu žádného z porovnávaných registrů. Testováním příznaků lze například zjistit, zda jsou čísla stejná (Z = 1) apod.

• Podobná je instrukce **CPC Rd,Rr**. Rozdíl je stanoven jako Rd - Rr - C (uvažuje se výpůjčka).

• Instrukce **CPI Rd,data8** porovnává obsah registru **Rd** s konstantou (číslo registru je v rozsahu 16 až 31).

13.11. Další instrukce

Do této skupiny byly zařazeny instrukce (viz tab. 13.12), které se nehodí do žádné z výše uvedených skupin.

• Instrukce **NOP** neprovádí žádnou operaci. Pouze zpozdí vykonávání programu o jeden hodinový impuls. Proto se často používá k časování. Pokud uvážíme hodinový kmitočet mikrořadiče 10 MHz, bude vykonání této instrukce trvat 100 ns.

• Instrukce **SLEEP** zvolí režim spánku (viz kapitolu 4.2 z KE 5/2006).

• Instrukce **WDR** nuluje obvod Watch-Dog (sledovač korektního běhu programu, viz kapitolu 14).

• Instrukce **BREAK** představuje podporu pro ladění, programátor ji nemůže přímo používat.

13.12. Assembler

Kromě instrukcí je možno používat konstrukce, které se formou zápisu velmi podobající instrukcím. Mluvíme pak o pseudoinstrukcích nebo direktivách assembleru.

Nejdříve se zaměříme na základní pojmy.

Symboly - jsou alfanumerickou reprezentací číselných nebo znakových konstant, adres apod. Pro zápis symbolu lze použít malé a velké znaky anglické abecedy (a až z, A až Z), číslice (0 až 9) a speciální znak (_). Pro vzájemné odlišení zápisu čísla od symbolu je nutné, aby symbol nezačínal číslicí!

Malá písmena jsou vnitřně převedena na velká, překladač nerozlišuje „velikost" písmen.

Některé symboly jsou vyhrazeny a proto nemohou být použity uživatelem. Jedná se o mnemokódy instrukcí (např. **MOV**), direktivy assembleru, funkce assembleru (například **LOW**) a rezervované operandy (např. R0 až R31).

Příklady platných symbolů:

```
PI
Serial_Port_Buffer
LOC_4096
```

Příklady neplatných symbolů:

```
1.PROMENNA      ;zacíná císlicí
ALFA#           ;# není platným znakem
MOV             ;mnemokód
LOW             ;operátor assembleru
DSEG            ;direktiva assembleru
```

Návěští - je speciální případ symbolu, řídí se tedy stejnými pravidly. Navíc je však každé návěští zakončeno dvojtečkou (:).

Návěští se obvykle používá k označení určitého místa v programu (kam je třeba provést skok) nebo pro „pojmenování" proměnné.

Označení místa v programu:

```
CEKEJ:  DEC R19    ;sniž R19
        BRNE CEKEJ ;cekej do vynulování R19
```

Definice proměnné s jejím pojmenováním (viz direktivu **BYTE**):

```
X:      .BYTE 1    ;jednobajtová promenná X
```

ASCII literály - literál je přímé uvedení hodnoty. Např. 56 je celočíselný literál.

Znakové ASCII literály se uzavírají mezi apostrofy (’), řetězcové ASCII literály se uzavírají mezi uvozovky (“).

Příklady ASCII literálů:

```
LDI R19,'m'                ;do R19 nahraj znak m
.EQU APOSTROF ='''         ;APOSTROF
.DB "ATtiny2313"           ;retezec
```

Komentář - je libovolná poznámka zapsaná do zdrojového textu. Komentář začíná středníkem (;) a končí koncem řádku. Komentáře assembler ignoruje, slouží pouze pro lepší orientaci v programu.

Lokační čítač programového segmentu PC - překladač uchovává lokační čítač (aktuální pozici) pro programový segment. Hodnotu lokačního čítače udává symbol **PC**.

Typický příklad použití lokačního čítače je při instrukcích skoků (v dané konstrukci se příznak Z musí vynulovat jedině zvnějšku pomocí přerušení):

```
BRNE PC          ;cekej dokud Z=0
```

nebo při určování délky dat:

```
TEXT:   .DB "ATtiny2313"   ;retezec
        .EQU DELKA=PC-TEXT ;a jeho délka
```

Čísla a operátory. Překladač podporuje zápis čísel ve čtyřech soustavách (se základy: 2, 8, 10 a 16). Výchozí je desítková soustava.

Pro určení soustavy, v níž je číslo zadáno, se používají předpony. Vše je zřejmé z tab. 13.13. Ve sloupci **Příklad zápisu** je uvedena desítková hodnota 250 zapsaná v různých číselných soustavách.

Dvojkovému číslu předchází sekvence **0b**.

Osmičkovému číslu předchází tzv. **vodicí nula**. Tato nula nemá význam pro zápis hodnoty, pouze sděluje, že číslo je zapsáno osmičkově a ne desítkově.

Desítkové hodnoty se zapisují běžným způsobem. Pozor, vodicí nuly nyní nesmíte používat, protože překladač by pak bral hodnotu zapsanou osmičkově. Například **010** má desítkovou hodnotu **8** a ne **10**, jak by se dalo očekávat.

Šestnáctkovému číslu předchází dolar (**$**) nebo sekvence **0x**.

Při zápisu čísla nesmí být použita mezera nebo tabulátor. Předpona může být zapsána velkým ale i malým písmenem (například **B** i **b**, **X** i **x**).

Pro sestavení výrazu, který lze vyhodnotit v době překladu, se používají symboly uvedené v tab 13.14. Prioritu operandů lze měnit pomocí závorek.

Pro sestavení výrazu, který lze vyhodnotit v době překladu, lze používat ještě funkce uvedené v tab 13.15. Výraz se zapisuje do závorek.

Příklady použití operátorů a funkcí:

```
HIGH($ABCDH)       ;vrátí $AB
LOW($ABCDH)        ;vrátí $CD
7*4                ;vrátí 28
7/4                ;vrátí 1
0b1000<<2          ;vrátí 0b100000
0b1010>>2          ;vrátí 0b10
10+5               ;vrátí 15
25-17              ;vrátí 8
-1                 ;vrátí 0b1111111111111111
                   ;(zobrazí se ve 2. doplnku)
7=4                ;vrátí 0
7>4                ;vrátí 1
0b1101&0b0101      ;vrátí 0b0101
0b1101|0b0101      ;vrátí 0b1101
0b1101^0b0101      ;vrátí 0b1000
```

.EQU - definice symbolu - slouží pro pojmenování literálu nebo pro zavedení nového jména pro stávající symbol.

Formát: `.EQU Symbol=výraz`

Příklad:

```
.EQU IO_OFFSET=$23
.EQU PORTA=IO_OFFSET+2

.CSEG
OUT PORTA,R2               ;zápis na port A
```

Tab 13.13. Číselné soustavy

Číselná soustava	Zápis	Číslice	Příklad zápisu
dvojková (binární)	předchází 0b	0, 1	0b11111010
osmičková (oktalová)	předchází 0	0 až 7	0372
desítková (decimální)	bez vodící 0	0 až 9	250
šestnáctková (hexadecimální)	předchází $ nebo 0x	0 až 9, A až F	$fa, 0xfa

Tab 13.14. Operátory assembleru

Symbol	Název	Popis	Priorita
!	logická negace	vrací 1, pokud je výraz 0; vrací 0 pokud je výraz různý od 0	14
~	bitová negace	vrací invertované bity výrazu	14
-	záporné znaménko	otočí znaménko čísla (pro 1 vrátí –1, pro –25 vrátí 25)	14
*	násobení	vynásobí dva operandy	13
/	dělení	podělí celočíselně dva operandy	13
+	sčítání	sečte dva operandy	12
-	odčítání	odečte dva operandy	12
<<	posuv vlevo	posune levý operand o tolik bitů vlevo, kolik je určeno pravým operandem	11
>>	posuv vpravo	posune levý operand o tolik bitů vpravo, kolik je určeno pravým operandem	11
<	menší než	je-li levý operand menší než pravý, vrátí 1 (jinak 0)	10
<=	menší nebo rovno	je-li levý operand menší nebo roven pravému, vrátí 1 (jinak 0)	10
>	větší než	je-li levý operand větší než pravý, vrátí 1 (jinak 0)	10
>=	větší nebo rovno	je-li levý operand větší nebo roven pravému, vrátí 1 (jinak 0)	10
==	rovno	jsou-li oba operandy shodné, vrátí 1 (jinak 0)	9
!=	nerovno	jsou-li oba operandy různé, vrátí 1 (jinak 0)	9
&	bitový součin	vrátí bitový součin obou operandů	8
|	bitový součet	vrátí bitový součet obou operandů	7
^	výlučný bitový součet	vrátí výlučný bitový součet obou operandů	6
&&	logický součin	vrátí logický součin obou operandů (jsou-li oba operandy různé od 0, vrátí 1)	5
||	logický součet	vrátí logický součet obou operandů (jsou-li oba operandy rovny 0, vráti 0)	4

Tab 13.15. Funkce assembleru

Funkce	Popis
LOW	vrátí dolní bajt výrazu
HIGH	vrátí druhý bajt výrazu
BYTE2	vrátí druhý bajt výrazu (jako HIGH)
BYTE3	vrátí třetí bajt výrazu
BYTE4	vrátí čtvrtý bajt výrazu
LWRD	vrátí bity 0 až 15 výrazu
HWRD	vrátí bity 16 až 31 výrazu
PAGE	vrátí bity 16 až 21 výrazu
EXP2	vrátí mocninu 2 výrazu (2^X)
LOG2	vrátí celočíselnou část výrazu log2 (logaritmus při základu 2)

.SET - nastavení hodnoty symbolu - má podobný účel jako **.EQU**. Rozdíl mezi **.EQU** a **.SET** je v tom, že symbol zavedený přes **.EQU** nemůže měnit svou hodnotu. Symbol zavedený přes **.SET** se může modifikovat.

Formát: `.SET Symbol=výraz`

Příklad:

```
.SET IO_OFFSET=$23
.SET PORTA=IO_OFFSET+2

.CSEG
LDI R19,$FF
OUT PORTA,R19              ;zápis na PA
```

Výběr segmentu. Jsou definovány 3 segmentové direktivy: **CSEG**, **DSEG** a **ESEG** pro výběr jednoho ze tří paměťových prostorů mikrořadičů AVR - viz tab. 13.16. Výchozím segmentem je **CSEG**.

Formát: `.CSEG|.DSEG|.ESEG`

Tyto direktivy definují počátek programového, datového nebo E^{2}PROM segmentu. Zdrojový soubor může obsahovat více segmentů jednoho typu, které se při překladu spojí do jediného cílového segmentu.

Všechny segmenty mají vlastní lokační čítače. V programovém segmentu se data umísťují po slovech (dvoubajtech), v datovém a E^{2}PROM segmentu se data umísťují po bajtech.

Tab 13.16. Výběr segmentu

Direktiva	Segment
.CSEG	programový segment (paměť programu)
.DSEG	datový segment (RAM)
.ESEG	segment E^{2}PROM

.BYTE - vyhrazení prostoru v bajtech - slouží k vyhrazení prostoru v datovém segmentu (RAM). Výraz udává počet bajtů, které chceme pro proměnnou vyhradit.

Formát: `[Náveští:] .BYTE výraz`

Lokační čítač se zvýší o hodnotu udanou výrazem (při překročení rozsahu segmentu je hlášena chyba).

Příklady:

```
          .EQU TAB_SIZE=10
          .DSEG              ;datový segment
VAR1:     .BYTE 1            ;vyhrad jeden bajt
TAB:      .BYTE TAB_SIZE     ;vyhrad TAB_SIZE bajtů

          .CSEG
          LDI R30,LOW(VAR1)  ;nastaví dolní bajt Z
          LDI R31,HIGH(VAR1) ;nastaví dolní bajt Z
          LD R1,Z            ;nahraje VAR1 do R1
```

.DB - uložení konstanty do paměti programu - slouží k uložení konstanty (v rozměru bajtu) do Flash nebo E^{2}PROM. Proto je tato direktiva použitelná pouze v segmentech **CSEG** a **ESEG**.

V případě programové paměti (Flash) se paměť přiděluje po celých slovech. Proto jsou dva po sobě následující bajty uloženy do jednoho slova. Je-li na jednom řádku uveden lichý počet bajtů, je dolní bajt posledního slova naplněn uvedenou hodnotou a horní bajt je vynulován. Číselné hodnoty lze uvádět v rozsahu -128 až 255.

Formát: `[Náveští:] .DB výraz[,výraz...]`

Příklady:

```
.CSEG
consts: .DB 0,255, 0b01010101, -128, $aa

.ESEG
const2: .DB 1,2,3
```

.DW - uložení konstanty do paměti programu - slouží k uložení konstanty (v rozměru slova) do Flash nebo E^{2}PROM. Proto je tato direktiva použitelná pouze v segmentech **CSEG** a **ESEG**.

Číselné hodnoty lze uvádět v rozsahu -32768 až 65535.

Formát: `[Náveští:] .DW výraz[,výraz...]`

Příklady:

```
.CSEG
consts: .DW 0,255,-128, $FA0C

.ESEG
const2: .DW 1000,2000,3000
```

.ORG - nastavení počátku segmentu - slouží pro určení hodnoty lokačního čítače v aktuálním segmentu.

Výchozí pozice lokačního čítače je pro segmenty **.CSEG** a **.ESEG** nula (0). Pro segment **.DSEG** je výchozí hodnotou $60 (přeskočí se registrové pole a vstupně/výstupní registry).

Formát: `.ORG výraz`

Příklady:

```
          .DSEG              ;Start datového seg.

          .ORG $80           ;Nastaví RAM
                             ;adresu na $80
VAR:      .BYTE 1            ;vyhradí bajt
                             ;na adrese $80
          .CSEG
          .ORG $10           ;PC=$10
          MOV R0,R1;         ;nejaká operace
```

.DEF - definice symbolického jména registru - umožňuje odkazovat se na registry pomocí nově zavedených symbolů. Každý registr může mít více takových nových jmen, symbol je možno redefinovat.

Formát: `.DEF Symbol=registr`

Příklady:

```
          .DEF TEMP=R16      ;R16 jako TEMP
          .DEF IOR=R0        ;R0 jako IOR

          .CSEG
          LDI TEMP,$F0       ;TEMP=$F0
          IN IOR,$3F         ;IOR=SREG
```

.INCLUDE - vložení obsahu externího souboru - vloží do zdrojového souboru obsah diskového souboru s udaným jménem. Tato technika je vhodná pro rozdělení programu do kratších částí nebo pro uložení často používaných podprogramů do zvláštního souboru. Také se takto vkládají soubory s definicí registrů mikrořadiče.

Formát: `.INCLUDE "jméno_souboru"`

.EXIT - konec programu - signalizuje konec zdrojového textu. Překlad zdrojového souboru je standardně ukončen koncem tohoto souboru. Použitím této direktivy lze překlad ukončit předčasně.

Formát: `.EXIT`

14. Další vlastnosti mikrořadiče ATtiny2313

V této kapitole budou vysvětleny další, dosud neuvedené, vlastnosti mikrořadiče ATtiny2313. Jedná se o popis systémového časování, resetovacího obvodu a propojek.

14.1. Systémové hodiny a volby synchronizace

Na obr. 14.1 je ukázána distribuce hodinového signálu v mikrořadiči ATtiny2313. Jednotlivé hodinové signály:

- **clk_{CPU}** - hodiny jádra procesoru; řídí časování AVR jádra (registry, SREG, SRAM, SPI),
- **$clk_{I/O}$** - hodiny vstupně/výstupních jednotek; řídí vstupně/výstupní jednotky (čítač/časovač, USI, USART), také se používá pro vnější přerušení,
- **clk_{FLASH}** - hodiny pro Flash; řídí operace nad rozhraním Flash (obvykle se aktivují současně s clkCPU).

Zdroje synchronizace

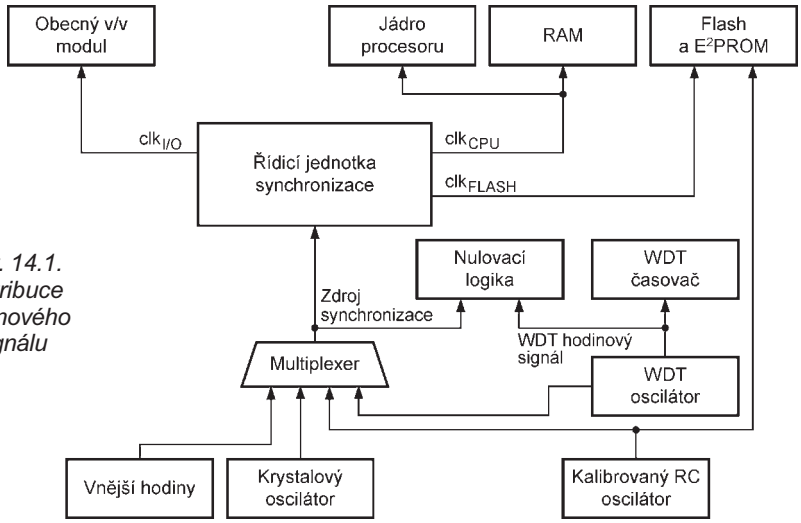
Synchronizační zdroj lze vybrat pomocí programovatelných propojek **CKSEL0 až CKSEL3** uložených ve Flash - viz tab. 14.1.

Když se jádro procesoru probouzí z režimů **Power-down** nebo **Power-save**, definuje vybraný synchronizační zdroj startovací čas (zajistí ustálení kmitů oscilátoru před vykonáním první instrukce). Podobně po resetu je vložen time-out (čekací interval), který zajišťuje stabilizaci spotřeby před vstupem do normálního režimu. Tento interval je stanoven v počtu cyklů zabudovaného WDT oscilátoru.

Obě tyto informace však závisí na zvoleném synchronizačním zdroji, konkrétní hodnoty jsou proto uvedeny dále.

Výchozí zdroj hodin

Mikrořadič ATtiny2313 je dodáván v konfiguraci propojek: **CKSEL = 0100**, **SUT = 10**, **CKDIV8** naprogramována. Takže je použit kalibrovaný vnitřní oscilátor *RC* a nejdelší možný interval rozběhu a time-outu. Pracovní kmitočet je pak 1 MHz. Jiné hodnoty propojek lze nastavit i pomocí ISP programátoru.



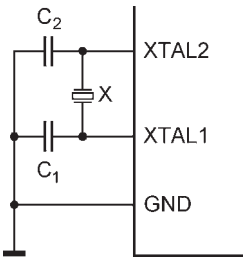
Obr. 14.1. Distribuce hodinového signálu

Tab. 14.1. Výběr synchronizačního zdroje

CLKSEL3 až CLKSEL0	Zdroj synchronizace
0000	vnější hodiny
0010	kalibrovaný vnitřní oscilátor *RC* 4 MHz
0100	kalibrovaný vnitřní oscilátor *RC* 8 MHz
0110	Watchdog oscilátor 128 MHz
1000 až 1111	vnější krystal/keramický rezonátor
0001, 0011, 0101, 0111	vyhrazeno (nepoužívat)

Tab. 14.2. Pracovní režimy krystalového oscilátoru

CLKSEL3 až CLKSEL0	Kmitočtový rozsah [MHz]	Doporuč. kapacita C1, C2 [pF]
100X	0,4 až 0,9	-
101X	0,9 až 3,0	12 až 22
110X	3,0 až 8,0	12 až 22
111X	8,0 a více	12 až 22



Obr. 14.2. Připojení krystalu k mikrořadiči. Volba kapacit C1 a C2 podle tab. 14.2

Krystalový oscilátor

Krystalový oscilátor souvisí s vývody XTAL1 a XTAL2, které představují vstup a výstup invertujícího zesilovače vestavěného krystalového oscilátoru (obr. 14.2). Pracovní režimy krystalového oscilátoru jsou uvedeny v tab. 14.2.

Propojka **CKSEL0** společně s propojkami **SUT1 a SUT0** volí startovací čas a interval nulovacího time-outu (viz tab. 14.3).

Kalibrovaný vnitřní oscilátor *RC*

Kalibrovaný vnitřní oscilátor *RC* poskytuje hodinový signál o nominálním kmitočtu **8 MHz**. Kalibrace je stanovena pro U_{CC} = 3 V a teplotu 25 °C; relativní odchylka je 10 %. Pomocí propojek CKSEL3 až CKSEL0 lze navolit i kmitočet 4 MHz (tab. 14.4) a naprogramováním propojky **CKDIV8** pak děličku osmi.

Startovací čas a interval nulovacího time-outu je uveden v tab. 14.5. Kalibrovaný oscilátor je také použit pro WDT oscilátor a nulovací time-out.

Kalibrace je zajištěna výrobcem pomocí kalibračního bajtu. Ten je stanoven po zapouzdření a vypálen do čipu. Při resetu se hodnota kalibračního bajtu nahraje do registru **OSCCAL** (obr. 14.3 na str. 33).

Vzhledem k tomu, že je registr určen nejen pro čtení, ale i pro zápis, lze kalibraci provedenou výrobcem měnit. Čím je obsah kalibračního registru vyšší, tím je vyšší i generovaný kmitočet - viz tab. 14.6.

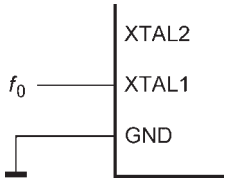
Pozor! Kalibrovaný oscilátor *RC* je používán i pro časování operací při přístupu k Flash nebo E²PROM. Pokud tedy plánujete zapisovat do těchto pamětí za běhu programu, nesmí se výchozí hodnota **OSCCAL** změnit o více než 10 %, jinak hrozí selhání zápisu.

Vnější hodiny

Pokud se rozhodnete používat jako zdroj synchronizace vnější hodinový signál f_0 (obr. 14.4), musíte nastavit propojky **CKSEL = 0000**.

Pozor! Stabilita kmitočtu horší než 2 % může vést k nedefinovanému chování mikrokontroléru!

Startovací čas a interval nulovacího time-outu jsou uvedeny v tab. 14.5.



Obr. 14.4. Připojení vnějšího zdroje hodinového kmitočtu f_0 k mikrořadiči

Obr. 14.3.
Registr **OSCCAL**

Bit	7	6	5	4	3	2	1	0
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	Kalibrační hodnota specifická pro konkrétní mikrokontrolér							

Tab. 14.3. Startovací čas a interval time-outu po resetu při použití krystalového oscilátoru (CK značí periodu hodinového cyklu)

CKSEL0	SUT1, SUT0	Startovací čas pro Power-down a Power-save	Nulovací time-out pro U_{CC} = 5 V	Doporučené použití
0	00	258 hodinových cyklů	14CK+4,1 ms	rezonátory, rychlý náběh jmenovitého odběru
0	01	258 hodinových cyklů	14CK+65 ms	rezonátory, pomalý náběh jmenovitého odběru
0	10	1024 hodinových cyklů	14CK	rezonátory, povolení detekce poklesu napětí
0	11	1024 hodinových cyklů	14CK+4,1 ms	rezonátory, rychlý náběh jmenovitého odběru
1	00	1024 hodinových cyklů	14CK+65 ms	rezonátory, pomalý náběh jmenovitého odběru
1	01	16384 hodinových cyklů	14CK	krystaly, povolení detekce poklesu napětí
1	10	16384 hodinových cyklů	14CK+4,1 ms	krystaly, rychlý náběh jmenovitého odběru
1	11	16384 hodinových cyklů	14CK+65 ms	krystaly, pomalý náběh jmenovitého odběru

Tab. 14.5. Startovací čas a interval time-outu po resetu (CK značí periodu hodin)

SUT1, SUT0	Startovací čas pro Power-down a Power-save	Nulovací time-out pro U_{CC} = 5 V	Doporučené použití
00	6 hodinových cyklů	14CK	povolení detekce poklesu napětí
01	6 hodinových cyklů	14CK+4,1 ms	rychlý náběh jmenovitého odběru
10	6 hodinových cyklů	14CK+65 ms	pomalý náběh jmenovitého odběru
11	vyhrazeno		

Tab. 14.4. Výběr kmitočtu odvozeného z kalibrovaného RC oscilátoru

CLKSEL3 až CLKSEL0	Zdroj synchronizace
0010	4 MHz
0100	8 MHz

Tab. 14.6. Kmitočtový rozsah vnitřního oscilátoru RC

Hodnota OSCCAL	Kmitočtový rozsah v procentech nominální hodnoty
$00	50 až 100%
$7F	75 až 150%
$FF	100 až 200%

Tab. 14.7. Předdělička hodin

CLKPS3 až CLKPS0	Dělicí poměr
0000	1
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1001 až 1111	vyhrazeno (nepoužívat)

Vnitřní oscilátor 128 kHz

Vnitřní oscilátor 128 kHz je oscilátor s malým příkonem, který poskytuje kmitočet 128 kHz (při U_{CC} = 3 V a teplotě 25 °C). Tento zdroj synchronizace je vybrán při **CKSEL = 0110**. Startovací čas a interval nulovacího time-outu jsou uvedeny v tab. 14.5.

Registr CLKPR - registr předděličky hodin

Pomocí registru **CLKPR** je možné dělit vnitřní kmitočet mikrořadiče v rozsahu 1 až 256 (viz obr. 14.5 a tab. 14.7).

Význam jednotlivých bitů:

• **CLKPCE (povolení změny předděličky hodin)** - pro povolení změny bitů **CLKPS** musí být do tohoto bitu zapsána 1. Změna bitu **CLKPCE** může proběhnout pouze tehdy, pokud současně do bitů **CLKPS** zapisujeme 0000. Bit **CLKPCE** je vynulován hardwarově po uplynutí 4 hodinových cyklů zápisu do **CLKPCE** nebo **CLKPS**.

• **CLKPS3 až 0 (bity pro nastavení předděličky)** - tyto bity volí dělicí poměr mezi vybraným hodinovým zdrojem a vnitřními hodinami. Aby se zabránilo nedefinované změně hodinového kmitočtu, musíme pro nastavení těchto bitů provést speciální proceduru, které sestává ze dvou kroků:

1) Do registru **CLKPR** zapíšeme hodnotu 0b1000 0000 (CLKPCE = 1, CLKPS3 až CLKPS0 = 0000).

2) V průběhu 4 hodinových cyklů zapíšeme do registru **CLKPR** požadované nastavení předděličky (CLKPCE musí být nyní vynulován, tedy 0b0000 ABCD).

V průběhu zápisu do registru **CLKPR** musí být zakázáno přerušení, jinak může být zapisovací procedura přerušena aktivací přerušení!

Propojka **CKDIV8** určuje výchozí hodnotu bitů **CLKPS**. Pokud je propojka **CKDIV8** naprogramována, je výchozí hodnota **CLKPS = 0011** (dělí osmi). Pokud propojka **CKDIV8** není naprogramována, je výchozí hodnota **CLKPS = 0000** (nedělí). Ostatní hodnoty jsou zřejmé z tab. 14.7.

14.2. Zdroje resetu

Mikrořadič ATtiny2313 má čtyři zdroje resetu:

• **Reset po připojení napájení** (Power-on Reset) nastane, pokud napájecí napětí překročí velikost U_{POT} (typicky 1,1 až 1,2 V).

• **Vnější reset** je aktivován vnějším vývodem **RESET/** (znak „/“ znamená negaci), na který je přivedena úroveň „log. 0“ po dobu alespoň 2,5 µs.

• **Watchdog reset** nastane po vypršení periody obvodu Watchdog (WDT), je-li tento obvod aktivován.

• **Reset při poklesu napájení** (Brown-out reset) nastane, pokud napájecí napětí poklesne pod hodnotu U_{BOT} (typicky 1,8 V pro BODLEVEL = 110 nebo 2,7 V pro BODLEVEL = 101 nebo 4,3 V pro BODLEVEL = 100).

V průběhu resetu jsou všechny vstupně/výstupní registry nastaveny na jejich výchozí hodnoty a program se odstartuje od adresy **$0000**. Obvykle se opět jedná o instrukci **RJMP**, která „zavede” vykonávání programu do inicializační rutiny.

Vstupně/výstupní vývody se okamžitě nastaví na jejich výchozí hodnoty (nezávisle na běhu oscilátoru).

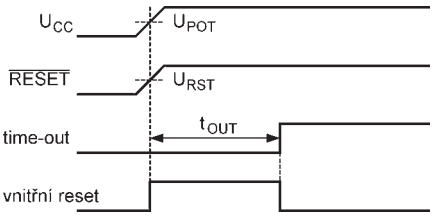
Po deaktivaci všech zdrojů resetu je připojeno zpoždění, které prodlouží vnitřní reset. To zajistí ustálení odběru před přechodem do normálního režimu. Interval tohoto přídavného zpoždění lze ovlivnit propojkou **CKSEL**.

Power-on Reset

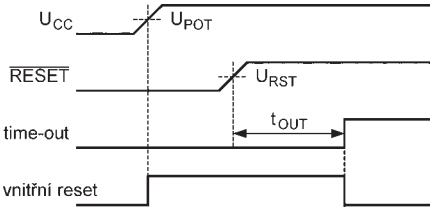
Power-on Reset je speciální obvod, který zajišťuje resetování mikrořadiče

Obr. 14.5.
Registr **CLKPR**

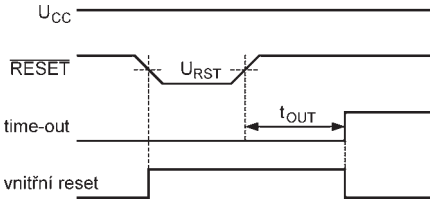
Bit	7	6	5	4	3	2	1	0
	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	dle nastavení propojky **CKDIV8**, viz popis			



Obr. 14.6. Rozběh mikrořadiče, RESET/ připojen na U_{CC}



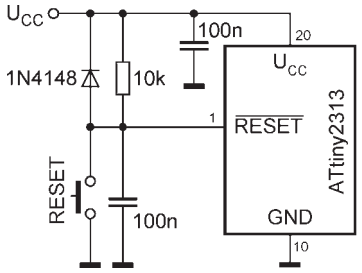
Obr. 14.7. Rozběh mikrořadiče při řízení vnějším signálem



Obr. 14.8. Průběh vnějšího resetu

po připojení napájení. Důležitou součástí tohoto obvodu je vnitřní časovač používající hodiny vytvářené vestavěným oscilátorem *RC*. Tento časovač blokuje start programu, dokud napájecí napětí (**U_{CC}**) zcela nenaběhne.

Pokud je použit tento vestavěný reset, je možné vývod **RESET/** připojit na U_{CC} přímo nebo přes vnější zvyšovací rezistor (pull-up) - viz obr. 14.6. Přidržením vývodu **RESET/** v úrovni „log. 0“ po náběhu U_{CC} lze nulovací periodu prodloužit (viz obr. 14.7).



Obr. 14.9. Jednoduchý resetovací obvod s článkem RC

Vnější reset

Vnější reset je generován úrovní „log. 0“ na vývodu **RESET/**. Pulsy delší než 2,5 µs generují reset i v případě, že krystalový oscilátor neběží. Po přechodu U_{RST} do úrovně „log. 1“ je vnitřní reset generován až do uplynutí doby ***t*OUT** (opět se uplatňuje vnitřní časovač řízený oscilátorem *RC* - viz obr. 14.8).

Praktická zapojení resetovacích obvodů jsou na obr. 14.9 a obr. 14.10. V obou případech se po připojení napájení začne nabíjet původně vybitý kondenzátor. Tím se drží vstup **RESET/** v úrovni „log. 0“. Pomocí tlačítka **RESET** je možné mikrokontrolér resetovat i manuálně.

Použití zvláštního obvodu **TL7705** podle obr. 14.10 je výhodnější, protože eliminuje případné překmity po připojení napájení. Cena tohoto obvodu je asi 20 Kč a jeho podrobnější popis je uveden v lit. [1].

WDT reset

Watchdog reset je generován po vypršení intervalu obvodu WDT. Tento impuls trvá jeden strojní cyklus (XTAL). Sestupná hrana tohoto pulsu odstartuje čekací periodu ***t*OUT** (opět se uplatňuje vnitřní časovač řízený oscilátorem *RC* - viz obr. 14.11).

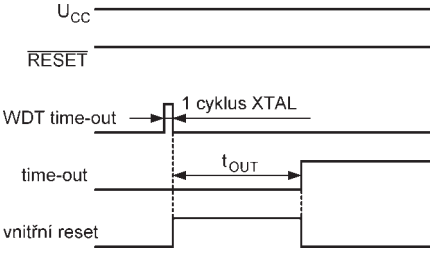
Watchdog je nezávislý čítač, který dohlíží na správný běh programu. Po jeho přetečení je vyvolán reset mikrořadiče.

Pokud tedy máme watchdog aktivován, musíme jej „čas od času” vynulovat, aby nepřetekl, protože pak by resetoval mikrokontrolér. To je normální stav. Když program (nedokonalostí našeho návrhu) „zabloudí” (např. uvízne v nekonečné smyčce, ve které se watchdog nenuluje), vyvede jej z „bloudění“ právě reset způsobený obvodem watchdog.

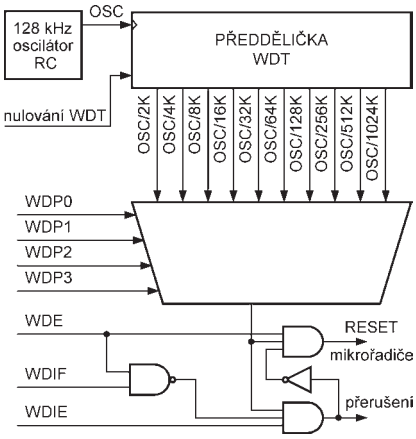
Watchdog čítá impulsy vestavěného oscilátoru RC (o kmitočtu zhruba 128 kHz). Interval, za který watchdog přeteče, je možné ovládat předděličkou (lze vybrat 10 různě dlouhých intervalů).

K nulování watchdogu slouží speciální instrukce **WDR** (WatchDog Reset).

Vypnutí již běžícího watchdogu je možné, ale musí se provést speciální sekvence instrukcí (to zajistí, že se neprovede náhodně, viz dále).



Obr. 14.11. Průběh WDT resetu Watchdog Timer (WDT)



Obr. 14.12. Blokové schéma obvodu watchdog

Pro ovládání watchodu slouží vstupně/výstupní registr **WDTCSR** (WatchDog Timer Control/Status Register). Viz obr. 14.13.

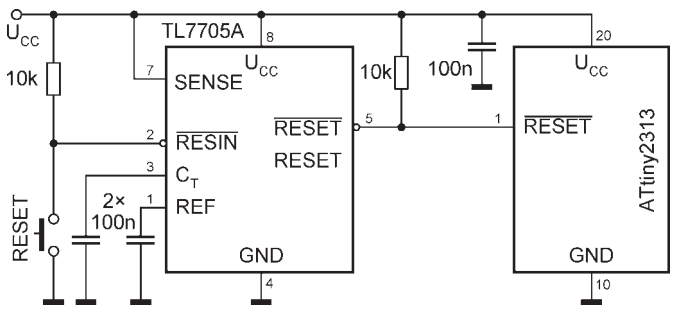
Význam jednotlivých bitů:

- **WDIF** - tento bit je nastaven po přetečení WDT v okamžiku, kdy je povoleno přerušení od WDT. Bit **WDIF** je nulován hardwarově provedením odpovídající rutiny přerušení. Alternativně lze bit **WDIF** vynulovat zápisem 1 do tohoto bitu.
- **WDIE** - povolení přerušení od WDT. Je-li **WDIE = 1**, je zvolen režim přerušení a systémového resetu - viz tab. 14.9.
- **WDCE -** povolení změny WDT předděličky. Jednou nastavený bit **WDCE** se automaticky vynuluje po uplynutí 4 hodinových cyklů.
- **WDE** - povolení WDT systémového resetu. Bit **WDE** je překryt bitem **WDRF** z registru **MCUSR**. To znamená, že **WDE** je vždy nastaven, když je nastaven **WDRF**. Pro nulování **WDE** musíme nejdříve vynulovat **WDRF**. Tato schopnost brání vícenásobnému resetu.
- **WDP3** až **WDP0** - určují interval přetečení watchdogu (vybírají signál z předděličky - viz tab. 14.10).

Vypnutí obvodu watchdog

WDT může být programově vypnut pouze v případě, že propojka **WDTON** není naprogramována.

Vypnutí obvodu watchdog se pak musí provést přesně podle níže uvede-



*Obr. 14.10. Použití obvodu **TL7705A** při resetu*

Bit	7	6	5	4	3	2	1	0
	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
Čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0	0	X	0	0	0

*Obr. 14.13. Registr **WDTCSR***

Bit	7	6	5	4	3	2	1	0
	–	–	–	–	WDRF	BORF	EXTRF	PORF
Čtení/zápis	R	R	R	R	R/W	R/W	R/W	R/W
Výchozí hodnota	0	0	0		viz popis			

*Obr. 14.15. Registr **MCUCSR***

ného postupu (viz datasheet mikrořadiče str. 39):

• vynulujeme WDT provedením instrukce **WDR**,

• v rámci jediné instrukce musíme nastavit **WDCE = 1** a **WDE = 1** (dokonce i když je **WDE = 1** nastaven před touto operací) a zároveň je třeba zachovat stav ostatních bitů registru **WDTCSR** (aby nenastala nedefinovaná změna nastavení předděličky),

• v průběhu následujících 4 strojních cyklů zapíšeme do **WDE = 0**.

Instrukce WDR a nulování watchdogu

Instrukci **WDR** je vhodno provést před aktivací watchdogu. Ze schématu zapojení (obr. 14.12) je totiž zřejmé, že watchog je průchozí a počítá impulsy ihned po resetu mikrokontroléru. Pokud bude obsah nenulový, bude první interval WDT time-out kratší než případné další intervaly.

Pokud ale provedeme instrukci **WDR** před aktivací watchdogu, bude obsah vynulován a time-out bude korektně odměřen.

Brown-out Reset

Mikrořadič ATtiny2313 má vestavěný detektor podpětí, který monitoruje úroveň napájecího napětí a porovnává ji s prahovou spouštěcí úrovní (obr. 14.14).

Spouštěcí úroveň lze nastavit na hodnotu U_{BOT} = 1,8 V (BODLEVEL = 110), U_{BOT} = 2,7 V (BODLEVEL = 101), U_{BOT} = = 4,3 V (BODLEVEL = 100). Pro BODLEVEL = 111 je detektor podpětí odstaven.

Detektor podpětí vykazuje hysterezi U_{HYST} asi 50 mV, aby se zabránilo aktivaci resetu při mírném poklesu napájení. Platí:

$$U_{BOT+} = U_{BOT} + (U_{HYST}/2) \text{ a}$$

$$U_{BOT-} = U_{BOT} - (U_{HYST}/2).$$

Je-li detektor zapnut, způsobí pokles U_{CC} pod hladinu U_{BOT-} okamžitou aktivaci resetu. Když se pak U_{CC} zvětší nad U_{BOT+}, odstartuje se zpožďovací interval, který prodlouží interval vnitřního resetu. Detektor uvažuje pouze takové poklesy napájecího napětí, které přesáhnou interval t_{BOD} = 2 µs.

Registr MCUSR

Mikrořadič ATtiny2313 disponuje stavovým registrem MCUSR (MicroController Unit Control and Status Register), který poskytuje informaci o zdroji, který vyvolal reset procesoru (obr. 14.15).

Význam jednotlivých bitů:

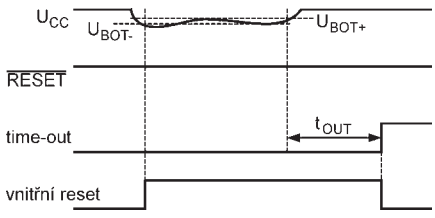
• **WDRF** (příznak WDT resetu) je nastaven po WDT resetu. Tento bit se nuluje při power-on resetu nebo zápisem 0 do tohoto bitu.

• **BORF** (příznak brown-out resetu) je nastaven po brown-out resetu. Tento bit se nuluje při power-on resetu nebo zápisem 0 do tohoto bitu.

• **EXTRF** (příznak vnějšího resetu) je nastaven po vnějším resetu (aktivaci vstupu **RESET/**). Tento bit se nuluje při power-on resetu nebo zápisem 0 do tohoto bitu.

• **PORF** (příznak power-on resetu) je nastaven po power-on resetu. Tento bit se nuluje pouze zápisem 0 do tohoto bitu.

Obsah registru **MCUCSR** je třeba číst co nejdříve po rozběhnutí programu. Pokud je registr vynulován před tím, než nastane jiná příčina resetu, nelze příčinu resetu pomocí těchto příznaků rozlišit.



Obr. 14.14. Reakce na pokles napětí

14.3. Propojky (fuses)

Mikrořadič ATtiny2313 disponuje tzv. propojkami (v originále fuses), které se programují společně s pamětí Flash pomocí programátoru (viz také obr. 2.15):

• **DWEN** - povolení funkce debugWIRE (ladění přímo v systému),

• **EESAVE** - obsah E^{2}PROM je zachován při smazání čipu,

• **SPIEN** - povolení sériového downloadu (programování přímo v aplikaci),

• **WDTON** - WDT je stále zapnut,

• **BODLEVE2 až BODLEVEL0** - volba referenční úrovně detektoru podpětí,

• **RSTDISBL** - odstavení vnějšího resetu (vstup **RESET/** je nefunkční),

• **CKDIV8** - dělení hodin osmi,

• **CKOUT** - výst. hodin na vývodu CKOUT (PD2),

• **SUT1, SUT0** - výběr startovací prodlevy,

• **CKSEL3 až CKSEL0** - výběr zdroje hodin.

Naprogramovaná propojka má hodnotu 0, nenaprogramovaná propojka pak hodnotu 1. Výchozí hodnoty propojek jsou uvedeny formou tab. 14.11.

Tab. 14.9. Konfigurace WDT

WDTON	WDE	WDIE	Režim	Akce po přetečení WDT
0	0	0	zastaven	žádná
0	0	1	přerušení	vyvolá přerušení
0	1	0	systémový reset	reset
0	1	1	přerušení a syst. reset	přerušení, potom přejde do resetu
1	X	X	systémový reset	reset

Tab 14.10. Výběr intervalu WDT

WDP3	WDP2	WDP1	WDP0	Předdělička	Typický WDT time-out, U_{CC} = 5 V
0	0	0	0	2K cykly	16 ms
0	0	0	1	4K cykly	32 ms
0	0	1	0	8K cyklů	64 ms
0	0	1	1	16K cyklů	0,125 s
0	1	0	0	32K cyklů	0,25 s
0	1	0	1	64K cyklů	0,5 s
0	1	1	0	128K cyklů	1 s
0	1	1	1	256K cyklů	2 s
1	0	0	0	512K cyklů	4 s
1	0	0	1	1024K cyklů	8 s
ostatní				–	vyhrazeno (nepoužívat)

Tab. 14.11. Výchozí hodnoty propojek (tovární nastavení)

Propojka	Hodnota	Význam
DWEN	1	debugWIRE odstaven
EESAVE	1	obsah E^{2}PROM není zachován
SPIEN	0	**sériový download povolen**
WDTON	1	WDT musí být zapnut programově
BODLEVEL2 až BODLEVEL1	111	detektor podpětí odstaven
RSTDISBL	1	**vnější reset povolen**
CKDIV8	0	**hodiny děleny osmi**
CKOUT	1	CKOUT odstaven
SUT1 až SUT0	10	dlouhá startovací prodleva
CKSEL3 až CKSEL0	0100	**hodiny z oscilátoru *RC* (8 MHz)**

15. Přípravek FT232TST a vývojový kit pro USB

Velká obliba rozhraní USB nás dovedla k tomu, že jako přílohu uvádíme konstrukci přípravku **FT232TST** (místo sériového portu s úrovněmi RS-232 nám poskytne emulovaný port na základě USB s TTL úrovněmi) a vývojový kit **USB2313KIT** (obdoba vývojového kitu SDK2313 s tím, že je ovládán a napájen z USB).

15.1. Přípravek FT232TST

Účelem přípravku **FT232TST** je emulace sériového portu pomocí USB. K tomu se používá známý obvod **FT232BM**. Výhodnou je zejména skutečnost, že výstupní úrovně jsou TTL **MAX232**). Další výhodou je také fakt, že novější počítače (především notebooky) již nedisponují sériovými porty.

Schéma testovacího přípravku (obr. 15.1) vychází z popisu obvodu **FT232BM**, který byl uveden v lit. [10]. Pro univerzální použití byly vývody UART rozhraní připojeny na konektor K2 (**PSL10**).

Rozložení jednotlivých vývodů na konektoru K2 odpovídá rozložení, které bylo zavedeno v knihách o mikrořadičích **ATMEL** (lit.[1] až [3]). Takže bude velmi snadné připojovat tyto mikrořadiče přímo k testovacímu přípravku!

Jumper J1 umožňuje odpojit napájecí napětí získané přípravkem přímo ze sběrnice USB. Pokud není na vývod *1* konektoru K2 na straně připojovaného zařízení přivedeno napětí, nechá se jumper zapojen (tak lze napájet přípravek přímo z portu). **Naopak, pokud má přípravek vlastní napájení, a to je na vývodu *1* konektoru K2, měl by být jumper vyjmut! To je náš případ.**

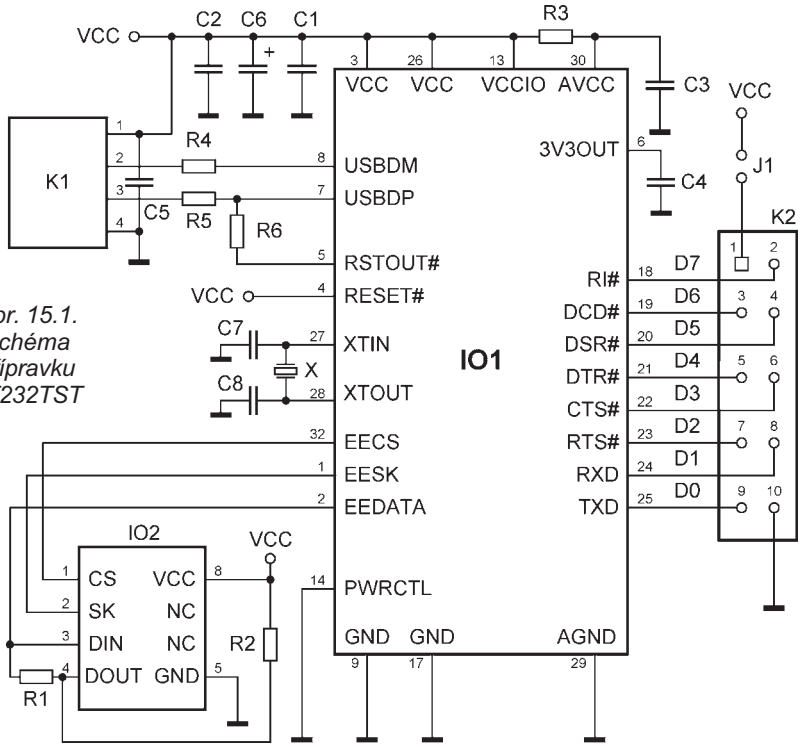
Vzhledem k tomu, že deska s plošnými spoji byla navržena jako jednostranná, byly použity součástky SMD. Vlastně to byla nutnost, protože sám obvod FT232BM je v provedení SMD. Součástky SMD se pájejí ze strany spojů. Strana součástek obsahuje pouze oba konektory (K1 pro připojení USB a K2 pro připojení řízeného zařízení), konfigurační E^{2}PROM v objímce, krystal 6 MHz, jednu drátovou propojku a jumper. Podklady pro zhotovení desky jsou na obr. 15.2 až obr. 15.4.

Seznam součástek pro FT232TST

(cena asi 230 Kč)

R1	2,2 kΩ, SMD 1206
R2	10 kΩ, SMD 1206
R3	470 Ω, SMD 1206
R4, R5	27 Ω, SMD 1206
R6	1,5 kΩ, SMD 1206
C1 až C3	100 nF/X7R, SMD 1206
C4	33 nF/X7R, SMD 1206
C5	10 nF/X7R, SMD 1206
C6	6,8 µF/10 V, tantal., SMD rozměr B
C7, C8	27 pF/NPO, SMD 1206
IO1	FT232BM
IO2	93LC46B–I/P
K1	USB1X90B PCB
K2	PSL10 (MLW10G)
X	krystal QM 6.000 MHz
J1	jumper

deska s plošnými spoji FT232TST



Obr. 15.1. Schéma přípravku FT232TST

15.2. Plug&Play ovladač

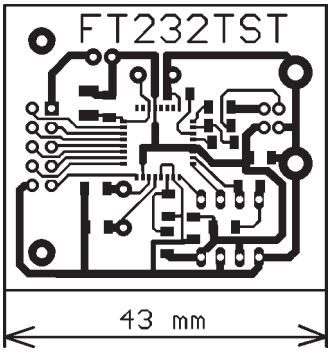
Po pečlivém osazení součástek můžeme přikročit k důležitému kroku, a to k připojení přípravku přes USB kabel A-B k počítači PC.

Pokud je vše v pořádku, detekuje operační systém nové zařízení (viz obr. 15.5). Po stisknutí tlačítka **Další** se zobrazí dialog podle obr. 15.6. Zvolíme možnost vlastní volby ovladače - operační systém může instalaci provést špatně. Potom pokračujeme tlačítkem **Další**.

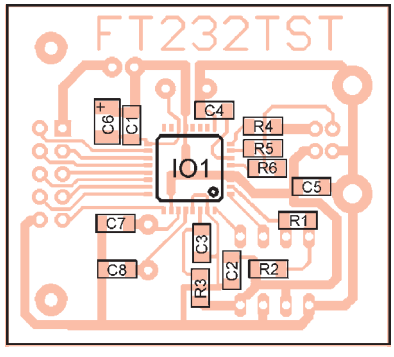
Nyní se operační systém zeptá na umístění souborů ovladače. Lze instalovat z diskety nebo zvolit adresář, ve kterém jsou soubory ovladače umístěny (viz obr. 15.7). Po stisknutí tlačítka **Další** se operační systém pokusí soubory najít. Ovladače jsou umístěny v [14] v adresáři **OVLADACE\VCP**.

V průběhu instalace se ještě zobrazí dialog podle obr. 15.8 (ovladač nemá certifikaci), který musíte potvrdit stisknutím tlačítka **Pokračovat**.

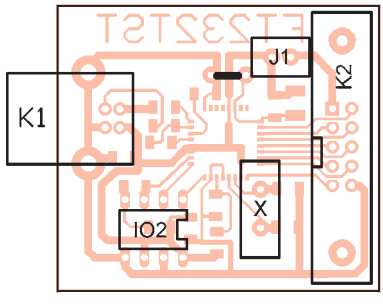
Pokud je vše v pořádku, dostaneme nakonec dialog podle obr. 15.9, který hlásí úspěšné dokončení instalace.



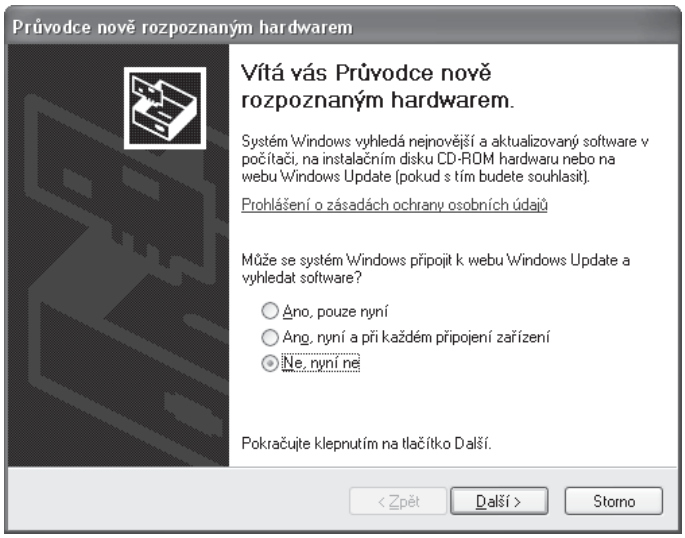
Obr. 15.2. Obrazec plošných spojů přípravku FT232TST (měř.: 1 : 1)



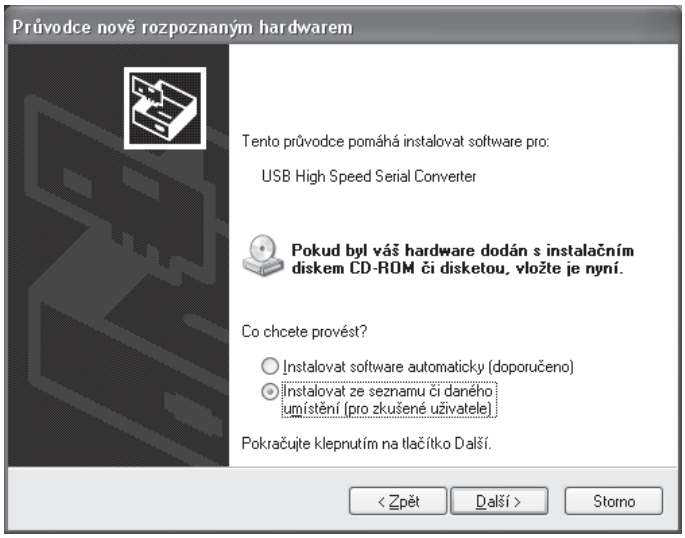
Obr. 15.3. Rozmístění součástek SMD na desce přípravku FT232TST



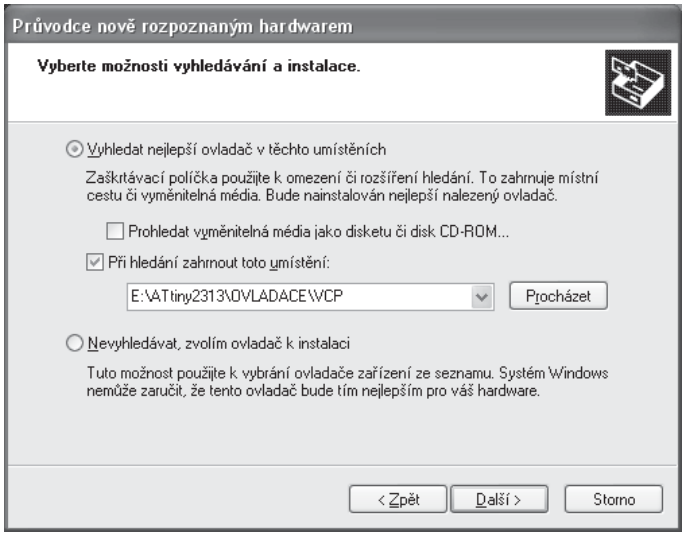
Obr. 15.4. Rozmístění vývodových součástek na desce FT232TST



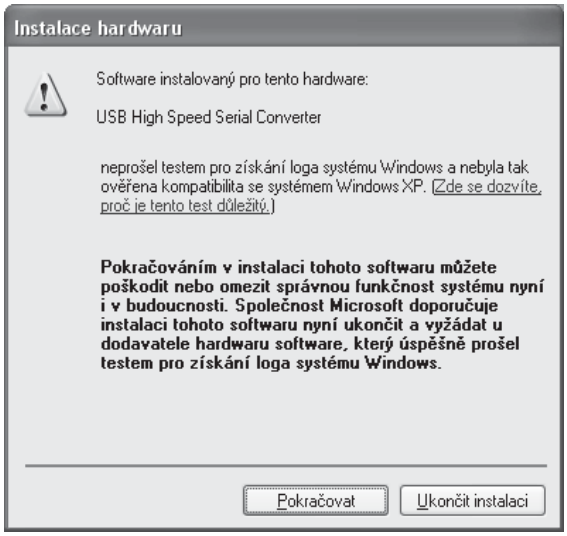
Obr. 15.5. Operační systém detekoval obvod FT232BM



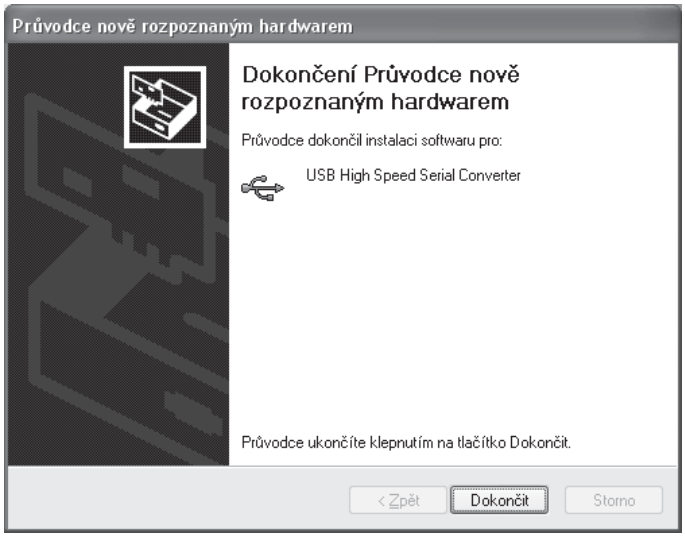
Obr. 15.6. Ovladač určíme sami!



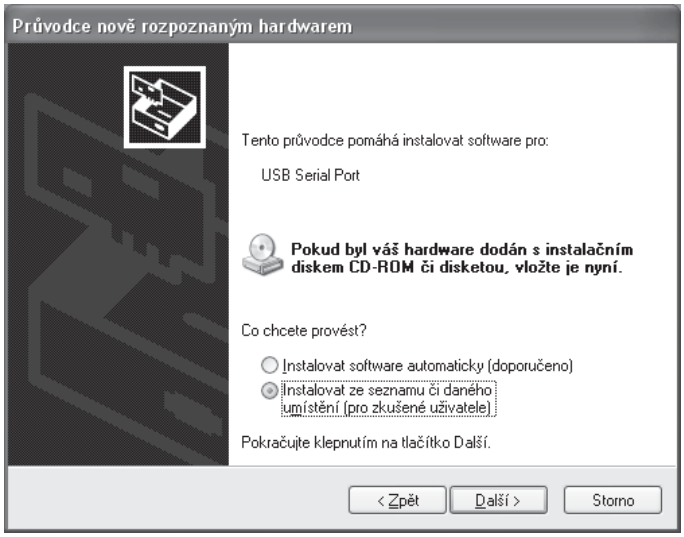
Obr. 15.7. Určení adresáře se soubory ovladače



Obr. 15.8. Instalace necertifikovaného ovladače



Obr. 15.9. Instalace je úspěšně dokončena



Obr. 15.10. Ovladač určíme sami!

Po stisknutí tlačítka **Dokončit** se ještě nainstaluje virtuální sériový port. Takže se objeví stejný dialog, který byl již uveden na obr. 15.5. Tlačítkem **Další** budeme pokračovat.

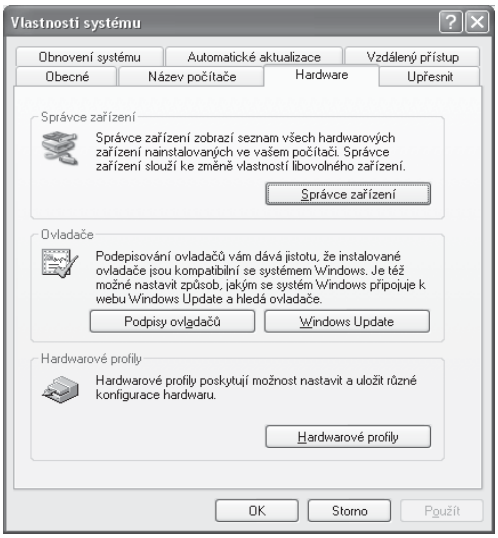
Poté je zobrazen dialog podle obr. 15.10. Srovnejte obrázky 15.6 a 15.10. Nyní se již neinstaluje zařízení „USB High Speed Serial Converter", ale virtuální zařízení „USB Serial Port". Opět vybereme instalaci vlastní a pokračuje stisknutím tlačítka **Další**.

Následně je zobrazen dialog podle obr. 15.7. Cesta k ovladači virtuálního sériového portu je stejná jako v případě instalace USB převodníku. Stisknutím tlačítka **Další** pokračujeme v instalaci.

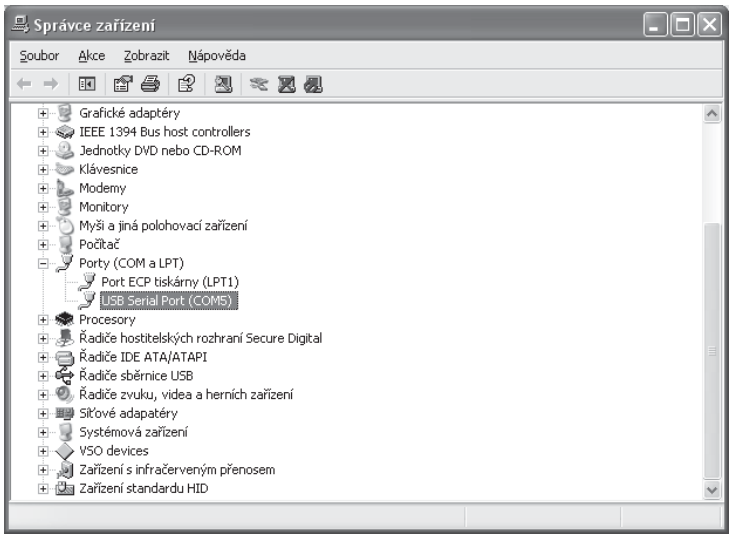
Situace s instalací se v mnohém opakuje. V průběhu instalace se opět objeví dialog podobný tomu, jako je uveden na obr. 15.8, který potvrdíme tlačítkem **Pokračovat**.

Instalace končí zobrazením podobného dialogu, jako je uveden na obr. 15.9.

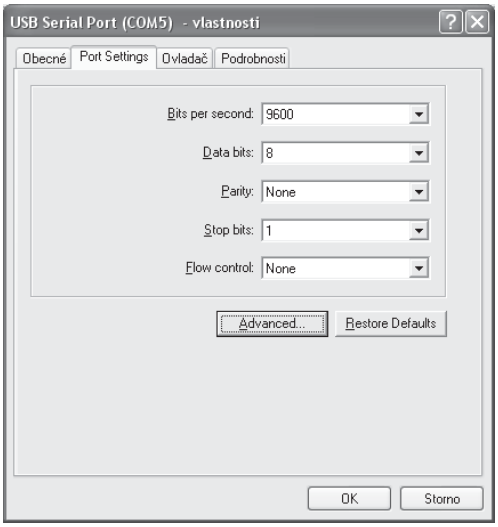
Po úspěšné instalaci se můžeme „podívat" do správce zařízení (**Start|Nastavení|Ovládací panely|Systém**). V dialogu dle obr. 15.11 vybereme kartu **Hardware** a stisknutím tlačítka **Správce zařízení** vyvoláme dialog podle obr. 15.12.



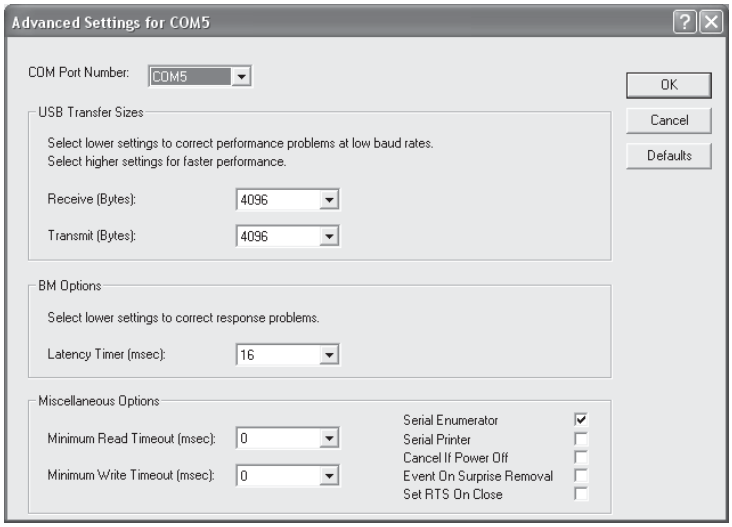
Obr. 15.11. Vyvolání dialogu Správce zařízení



Obr. 15.12. Zařízení je zaregistrováno v systému



Obr. 15.13. Dialog vlastností sériového portu



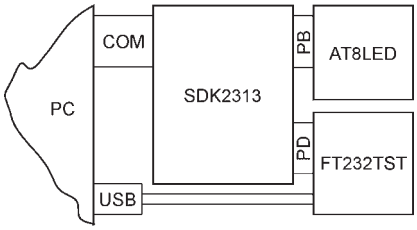
Obr. 15.14. Úprava čísla sériového portu

Ve **Správci zařízení** (obr. 15.12) rozbalíme složku **Porty (COM a LPT)** a najdeme položku s označením **USB Serial Port** (závorce je uvedeno číslo, které má port přiřazen).

Poklepáním na danou položku je zobrazen dialog vlastností sériového portu, který je uveden na obr. 15.13. Pokud chcete upravit číslo sériového portu, stiskněte tlačítko **Advanced**.

Úpravu čísla sériového portu provedeme pomocí dialogu dle obr. 15.14. Číslo portu se vybere v kombu označeném **COM Port Number**.

Pokud nyní chceme vyzkoušet funkčnost převodníku, můžeme to nejsnáze dokumentovat na příkladu, který byl označen jako **PROG_19** (viz kapitolu 10.11) Místo klasického sériového portu a přípravku **ATRS232+** připojíme port D mikrořadiče přes převodník **FT232TST** přímo na USB - viz obr. 15.15.



Obr. 15.15. Náhrada klasického sériového portu a přípravku ATRS232+ převodníkem FT232TST

15.3. Vývojový kit USB2313KIT

Přípravek **USB2313KIT** je vyspělý vývojový kit pro mikrořadič ATtiny2313, který má následující vlastnosti:

- malé rozměry,
- připojen na USB (není nutný sériový port),
- napájí se přes USB (není nutný napájecí zdroj),
- dražší a složitější konstrukce.

Z uvedených vlastností je zřejmé, že přípravek **USB2313KIT** má řadu výhod.

Není ovšem určen začínajícím, neboť obsahuje součástky SMD. Cena převodníku FT232BM pak také výrazně navyšuje cenu vývojového kitu. Proto byl v úvodu popisu mikrořadiče ATtiny2313 (v KE 5/2006) publikován konstrukčně levnější a jednodušší vývojový kit **SDK2313**.

Schéma vývojového kitu **USB2313KIT** je na obr. 15.16. Jádrem zapojení je konvertor **FT232BM** (IO1), který převádí signály sběrnice USB na RS-232 C), a mikrořadič **AT89S52-24AI** (IO3), který převádí signály RS-232 C na sběrnici SPI, používanou pro programování mikrořadiče ATtiny2313.

Paměť **93LC46B-I/P** (IO2) obsahuje VID a PID identifikátory nutné pro identifikaci vývojového kitu přes USB. Tato pamět i řídicí mikrořadič IO3 se naprogramují pomocí speciálních programů přímo v kitu, není nutný žádný zvláštní programátor.

Signál **SCK** pro programování IO4 musel být proudově posílen, pro jednoduchost je použit neinvertující zesilovač s tranzistory T2 a T3.

Vývody portu jsou vyvedeny na konektor **PB a PD**. Linky PD6 a PD7 jsou na konektoru PD spojeny, protože vývod **PD6** není na mikrořadiči IO4 k dispozici.

Tranzistor T1 spolu s tlumivkou L1, polyswitchem POL1 a kondenzátorem C11 zajišťují vyhlazené napájecí napětí asi 5 V/0,5 A. LED D1 indikuje úspěšné připojení kitu k počítači a pochopitelně i přítomnost napájecího napětí.

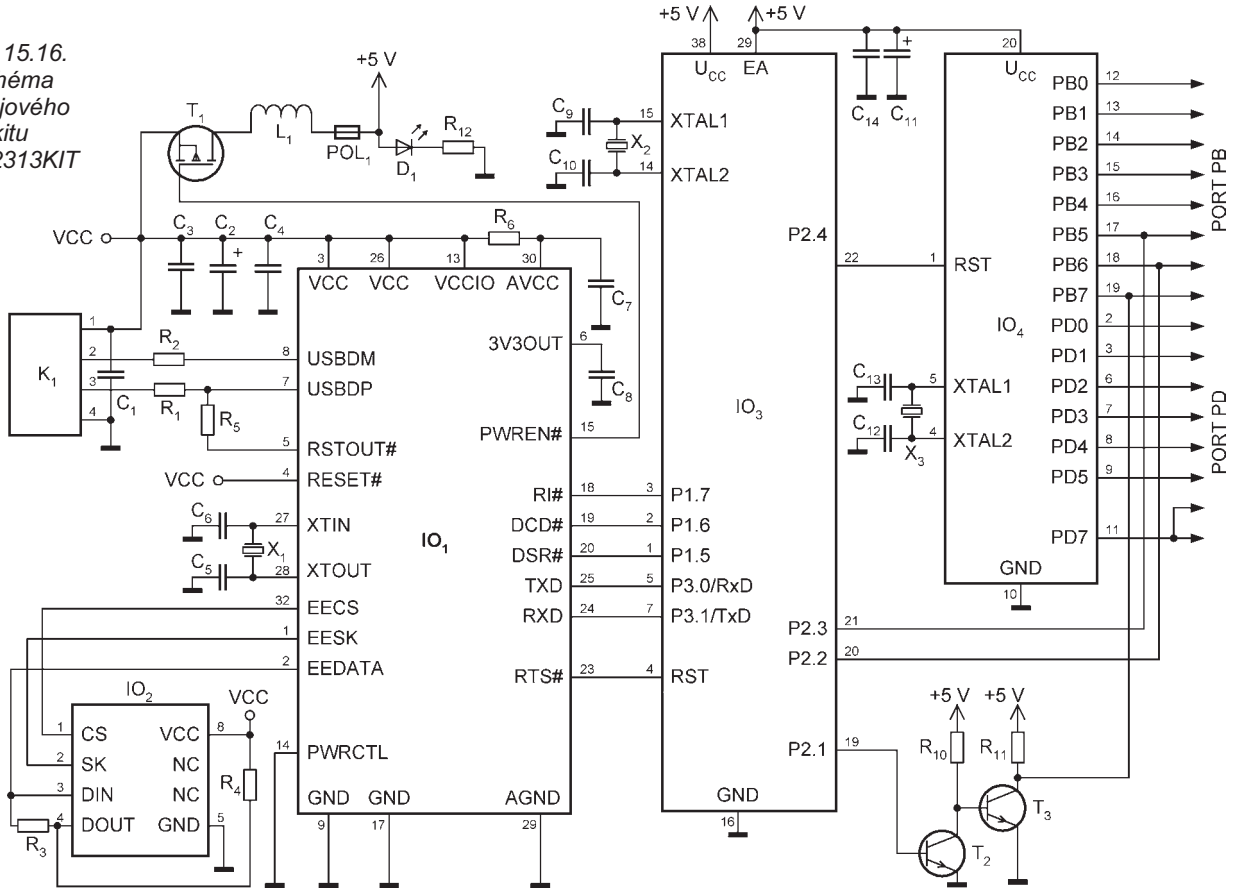
Komentář k zapojení IO1 je vynechán, více informací najdete např. v [10].

Podklady pro zhotovení desky s plošnými spoji jsou na obr. 15.17 až obr. 15.19. Desku je nejlepší zhotovit fotocestou. Při výrobě dbejte na to, aby se nespojily sousední vodivé cesty.

Při pájení dejte pozor zejména na IO1 a IO3. Nesmíte je pootočit! Na straně součástek je nutné osadit také 9 drátových propojek.

Pájecí špičky konektorů PB a PD doporučujeme nastavit (prodloužit) pomocí konektorových kolíků S2G10 tak, aby se zvětšila vzdálenost těchto ko-

Obr. 15.16. Schéma vývojového kitu USB2313KIT



nektorů od desky (viz obr. 15.20). Deska byla totiž navržena tak, aby ji bylo možné umístit do skříňky U-KM22. Do čela skříňky musíme pouze zhotovit díry pro konektory PB, PD a K1.

Seznam součástek pro USB2313KIT

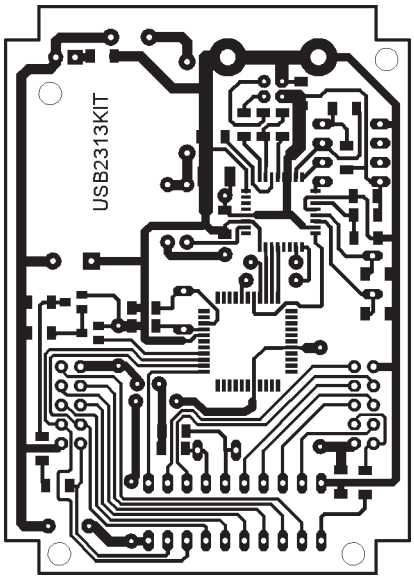
(cena asi 350 Kč)

R1, R2	27 Ω, SMD 1206
R3	2,2 kΩ, SMD 1206
R4, R10, R11	10 kΩ, SMD 1206
R5	1,5 kΩ, SMD 1206
R6, R12	470 Ω, SMD 1206
R7 až R9	0 Ω, SMD 1206
C1	10 nF/X7R, SMD 1206
C2	6,8 µF/10 V, tantal., SMD rozměr B
C3, C4, C7, C14	100 nF/X7R, SMD 1206
C5, C6, C9, C10	27 pF/NPO, SMD 1206
C8	33 nF/X7R, SMD 1206
C11	470 µF/25 V, radiální
C12, C13	15 pF/NPO, SMD 1206
L1	TL. 33 µH
X1	krystal QM 6,000 MHz
X2	krystal QM 24,000 MHz
X3	krystal QM 10,000 MHz
D1	LED 5 mm, 200 mcd
T1	IRFD9120
T2, T3	BC848
IO1	FT232BM
IO2	93LC46B-I/P
IO3	AT89S52-24AI
IO4	ATtiny2313-20PI
K1	USB1X90B PCB
PB, PD	MLW10G + nástavec S2G10
POL1	vratná pojistka RXE050

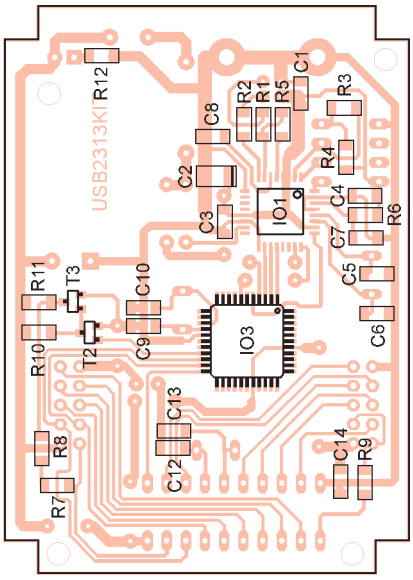
deska s plošnými spoji USB2313KIT

Před oživováním důrazně doporučujeme zkontrolovat správnost osazení jednotlivých součástek. Patrně nejdůležitější je zjistit, zda není zkrat mezi jednotlivými vývody konektoru USB.

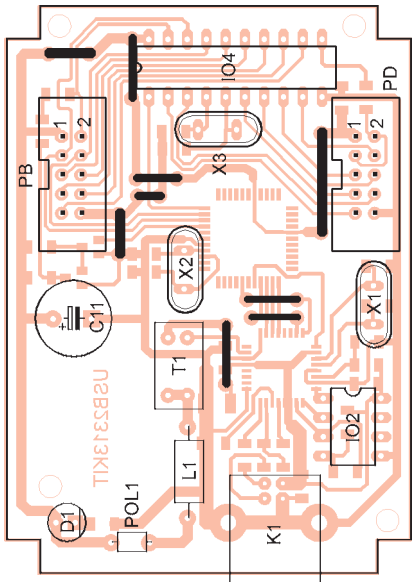
K oživení postačí běžný multimetr.



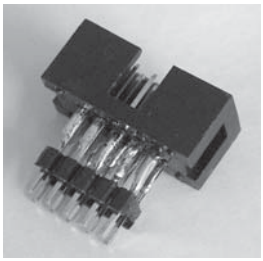
Obr. 15.17. Obrazec spojů na desce vývojového kitu USB2313KIT (měř.: 1 : 1, delší rozměr desky je 75 mm)



Obr. 15.18. Rozmístění součástek SMD na desce vývojového kitu USB2313KIT



Obr. 15.19. Rozmístění vývodových součástek na desce vývojového kitu USB2313KIT



Obr. 15.20. Prodloužení vývodů konektorů PB a PD pomocí konektorových kolíku S2G10

Instalace ovladačů

Po předchozí kontrole připojíme vývojový kit k počítači běžným USB kabelem typu A-B. Je-li vše v pořádku, detekuje systém nový hardware a vyzve nás k instalaci ovladačů. V této fázi je nutné použít pro instalaci ovladače dodávané ze [14], viz obr. 15.5 a 15.6. (nemůžeme se spoléhat např. na to, že Windows XP dokážou stáhnout ovladače z internetu). Ovladače najdete v [14] v adresáři **OVLADACE\DEBUG**.

Po úspěšné instalaci se rozsvítí LED D1 a mezi vývody *10* a *20* IO4 musí být napětí zhruba 5 V.

Pokud systém nový hardware nedetekuje, bude chyba v kabelu nebo v zapojení kolem obvodů IO1 a IO2. Doporučujeme zejména zkontrolovat kmitání krystalu X1 (digitální multimetr musí naměřit asi 2,5 V na libovolném z vývodů krystalu proti zemi) a přítomnost referenčního napětí 3,3 V na vývodu 3V3OUT (vývod *6* IO1).

Naprogramování konfigurační paměti

Konfigurační paměť IO2 lze naprogramovat bez nutnosti použít speciální programátor. Paměť lze programovat přímo v desce vývojového kitu pomocí programu **PGENER.EXE**, který je k dispozici v [14] v adresáři **DEBUG\USB2313KIT**.

Po spuštění programu stiskneme tlačítko **Generuj sériové číslo**. Toto tlačítko slouží pro zápis USB deskriptoru do paměti IO2 (typ 93LC46B). Bez platného deskriptoru nemůže být vývojový kit správně identifikován ovládacím programem. Sériové číslo je odvozeno od aktuálního data a času. Jméno zařízení je **USB2313KIT**.

Instalace nových ovladačů

Po předchozí operaci je nutné vývojový kit odpojit od USB a poté znovu připojit. Jedině tak se změny obsahu paměti promítnou do systému.

Obr. 15.21. Program PGENER v akci

Systém bude nyní vyžadovat nové ovladače (viz obr. 15.5 a 15.6), najdete je v [14] v adresáři **OVLADACE\USB2313KIT**.

Naprogramování řídicího mikrokontroléru

Řídicí mikrokontrolér IO3 naprogramujeme přímo v desce vývojového kitu pomocí programu **USBLOADER.EXE**, který je k dispozici v [14] v adresáři **DEBUG\USB2313KIT**.

Po spuštění programu vybereme jméno zařízení USB2313KIT, potom pomocí tlačítka se třemi tečkami (...) vybereme soubor pro download. Jedná se o soubor USB2313KIT.BIN, který je umístěn v [14] v adresáři **DEBUG\USB2313KIT**.

Potom zmáčkneme tlačítko **Spusť download**. Program nás informuje o průběhu operace.

Ovládací program

Ovládací program najdete v [14] v adresáři **USB2313KIT**. Po spuštění, je-li vše v pořádku, se zobrazí okno podle obr. 15.23.

Ovládání a nabídky spouštěné jednotlivými tlačítky mají stejný význam jako u dříve popsaného vývojového kitu **SDK2313** (viz kap. 2.3 v KE 5/2006).

Literatura

[1] *Matoušek, D.*: Práce s mikrokontroléry ATMEL AT89C2051, BEN, Praha 2002.

[2] *Matoušek, D.*: Práce s mikrokontroléry ATMEL AT89C8252, BEN, Praha 2002.

[3] *Matoušek, D.*: Práce s mikrokontroléry ATMEL AVR, BEN, Praha 2003.

[4] www.atmel.com, www stránky společnosti ATMEL.

[5] *Janeček, J.*: Projektování mikropočítačových systémů, skripta ČVUT Praha, 1995.

[6] *Jedlička, P.*: Přehled obvodů řady TTL 7400 I. a II. díl, BEN, Praha 1997.

[7] *Jedlička, P.*: Přehled obvodů řady CMOS 4000 I. a II. díl, BEN, Praha 2000.

[8] Data on Disc, katalog součástek SGS-Thomson na CD ROM, 1997.

[9] *Matoušek, D.*: Číslicová technika, BEN, Praha 2001.

[10] *Matoušek, D.*: USB prakticky s obvody FTDI 1. díl, BEN, Praha 2003.

[11] *Matoušek, D.*: Inteligentní LCD displeje, BEN, Praha 2004.

[12] *Matoušek, D.*: Udělejte si z PC v Delphi, 1. díl, BEN, Praha 2003.

[13] *Matoušek, D.*: Udělejte si z PC, 2. díl, BEN, Praha 2002.

[14] **Doprovodné CD-ROM k tomuto článku** - lze objednat u autora nebo jeho obsah stáhnout z dále uvedených webových stránek.

[15] *Matoušek, D.*: Práce s mikrokontroléry ATMEL ATmega16, BEN, Praha 2006.

[16] *Matoušek, D.*: Práce s mikrokontroléry ATMEL AT89LP2052, BEN, Praha 2006.

Informace na WWW

Vzhledem k tomu, že článek vychází ve formě časopisu, není jeho součástí doprovodné CD-ROM. Soubory, ze kterých lze CD-ROM vypálit, najdete na: *www.vspji.cz/matousek.php*

Druhou možností je objednat si doprovodný CD-ROM od autora (poštou na dobírku, cena včetně poštovného je 100 Kč), a to buď pomocí e-mailu: *matousekd@quick.cz*

nebo písemně na adrese:

Ing. David Matoušek
Vysoká škola polytechnická
Tolstého 16
586 01 Jihlava

Pro lepší představu připojuji krátké anotace některých doporučených knih.

Práce s mikrokontroléry ATMEL AVR

Kniha je zaměřena na popis mikrořadičů AVR: AT90S1200, AT90S2313, AT90S2343, AT90S4433, AT90S8515 a AT90S8535. Jsou zde publikovány vývojové kity kombinované s programátorem.

V knize najdete příklady zaměřené na použití čítačů/časovačů, jednotek Watch-Dog, Input Capture, Output Compare, PWM. Najdete zde i příklady realizace A/D a D/A převodníků (např. MCP3002); použití LCD, SPI sběrnice, sériového kanálu. Je uveden i příklad použití teplotního čidla SMT160-30.

USB prakticky s obvody FTDI

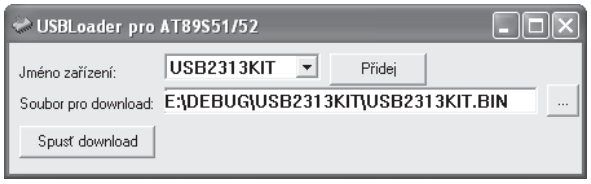
Kniha se věnuje popisu a praktickému použití obvodu FT232BM, obvod pracuje jako konvertor signálů sběrnice USB na signály asynchronního sériového kanálu včetně linek modemu.

V knize je publikováno mnoho konstrukcí s tímto obvodem: programátor AT89C2051, vývojové kity pro AT90S2313, zdroj s proudovým omezením řízený počítačem, měřicí deska pro USB, konvertory USB/RS-232.

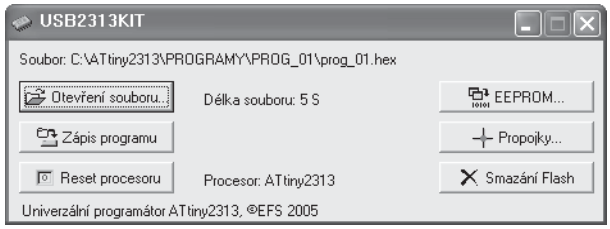
USB - měření, řízení a regulace pomocí sběrnice *USB*

Kniha zpřístupňuje USB i pro poloprofesionální aplikace. Krok za krokem jsou čtenáři seznamováni s vývojem HW i SW vhodného pro USB.

Na příkladech je uvedena práce s mikrořadiči CY7C63000 a AN2131 a jsou položeny i nezbytné základy programování na straně PC s jazyky Visual Basic a Delphi.



Obr. 15.22. Správné nastavení programu USBLoader



Obr. 15.23. Program USB2313KIT v akci